UNIVERSITETET I BERGEN

Institutt for lingvistiske, litterære og estetiske studium

DASP 350

Masteroppgave i datalingvistikk og språkteknologi

Høstsemester 2016

Towards the automatic evaluation of stylistic quality of natural texts: constructing a special-purpose corpus of stylistic edits from the Wikipedia revision history

Alexandr Kotlyarov

Abstract

This thesis proposes an approach to automatic evaluation of the stylistic quality of natural texts through data-driven methods of Natural Language Processing. Advantages of data driven methods and their dependency on the size of training data are discussed. Also the advantages of using Wikipedia as a source for textual data mining are presented. The method in this project crucially involves a program for quick automatic extraction of sentences edited by users from the Wikipedia Revision History. The resulting edits have been compiled in a large-scale corpus of examples of stylistic editing. The complete modular structure of the extraction program is described and its performance is analyzed. Furthermore, the need to separate stylistic edits stylistic edits from factual ones is discussed and a number of Machine Learning classification algorithms for this task are proposed and tested. The program developed in this project was able to process approximately 10% of the whole Russian Wikipedia Revision history (200 gigabytes of textual data) in one month, resulting in the extraction of more than two millions of user edits. The best algorithm for the classification of edits into factual and stylistic ones achieved 86.2% cross-validation accuracy, which is comparable with state-of-the-art performance of similar models described in published papers.

Sammendrag

Denne oppgaven foreslår en tilnærming til automatisk evaluering av stilistisk kvalitet av autentiske tekster gjennom datadrevne metoder for naturlig språkprosessering. Fordeler av datadrevne metoder og deres avhengighet av størrelsen på treningsdata blir diskutert. Også fordelene av bruk av Wikipedia som kilde for tekstuell datautvinning blir presentert. Metoden i dette prosjektet er basert på et program for hurtig automatisk ekstrahering av brukerredigerte setninger fra Wikipedias endringshistorikk. De ekstraherte endringene ble samlet i et stort korpus av eksempler på stilistisk redigering. Den fullstendige modulære strukturen av ekstraheringsprogrammet er beskrevet og ytelsen er analysert. I tillegg blir behovet for å skille mellom stilistiske endringer og faktiske endringer diskutert og en rekke klassifiseringsalgoritmer basert på maskinlæring blir foreslått for denne oppgaven. Programmet som ble utviklet i dette prosjektet klarte å prosessere ca. 10% av hele den russiske Wikipedias endringshistorikk (200 gigabytes av tekstdata) i én måned, noe som resulterte i ekstrahering av flere enn to millioner tekstendringer av brukere. Den beste algoritmen for å klassifisere endringer i stilistiske og faktiske endringer oppnådde 82% nøyaktighet, noe som er sammenlignbart med ytelsen i lignende spissteknologiske modeller i faglitteraturen.

Acknowledgements

First, I wish to thank Professor Koenraad De Smedt for being an incredible supervisor and generously providing time, support, guidance and a lot of patience throughout the preparation and review of this thesis.

I also want to thank my wife Olga for her support, help with Java programs, and relentless but constructive critique of my coding style.

I thank the University of Bergen for this wonderful opportunity to pursue a degree in computational linguistics.

And finally, I would like to thank Andrew Ng, Emily Fox, Carlos Guestrin and all other university teachers and industry specialists who put their time and effort into creation of online open-access courses – without you I wouldn't be able to learn about Python and Machine Learning as much as it was needed in order to finish this project – thank you.

Contents

Chapter 1

Introduction

1.1 Introduction

Wikipedia is a rich source of texts which can be studied – and indeed have been studied – from many perspectives. In this work I describe the benefits of the Wikipedia as a source of textual information and in particular explore the textual changes through editing as the specific objects of research. More generally, the goal of this work is to apply theories and methods of modern data-driven research in order to distill information from the edits and show how this information can be explored, analyzed and reused for other purposes.

1.2 Data-driven modeling

As an independent field Corpus Linguistics emerged in the middle of the twentieth century after the harsh criticism of the wide usage of *ad hoc* examples in linguistic works in 'Towards a description of English Usage' by Randolph Quirk [1]. Concerns expressed in this paper led to the creation of The Survey of English Usage group, the Brown Corpus, the Lancaster-Oslo-Bergen Corpus of British English. In this context, *top-down* model-driven corpus usage and annotation was gradually accommodated: the annotation of real data is done, manually or automatically, according to the a priori constructed theoretical framework to test how well the model describes the data. Here the corpus is not a source of linguistic knowledge but rather a performance measurement tool for theoretically constructed models, which is used to check them and improve based on the feedback [2].

With increased usage of computers in Corpus Linguistics and the development of various tools for statistical analysis of texts as well as the rise of statistical language modelling in the early 1990s, the opposite approach to handling corpora was proposed. This data-driven, or *bottom-up* principle advocates the inability of any human to correctly and fully model a language based only on their understanding of grammaticality. This principle considers the *top-down* approach useless, as it is not fully based on language data [3]. The *bottom-up* principle proposes the extraction of linguistic knowledge through statistical or algorithmic analysis of either completely unannotated corpora or corpora with minimal annotations (like Part of Speech (POS) tagging or Word Sense Disambiguation). The principle advises against any human correction of applied automatic annotation as to preserve the integrity and homogeneity of corpus data [3].

With the ever growing reliance of linguistics on computers and computational methods in the following decades, the data-driven approach became increasingly dominant, as the more and more theoretical and applied fields within the Natural Language Processing (NLP) acquired statistical and later neural methodologies, and became increasingly dependent on

large corpora and language datasets (which basically are just unannotated corpora by definition) [4].

1.3 High performance and training set size dependency of data-driven models

One of the main reasons for the success of the data-driven approach is in the reported performance superiority of statistical and neural NLP applications over linguistically motivated model-driven ones [6], [7], combined with the fact that the development of data-driven applications for NLP requires much less linguistic expertise from the developer. Commenting on the matter, Kyunghyun Cho states that:
“This lack of necessity for linguistic knowledge is not new. In fact, the most widely studied and used machine translation approach, which is (count-based) statistical machine translation, does not require any prior knowledge about source and target languages. All it needs is a large corpus” [8, p. 103].

However, the performance gain of data-driven models over rule-based ones comes with a property which could be both a blessing and a curse: extreme dependency on the size of the corpus used as the training set. The quite famous picture from Banko and Brill [9] presented on ***Figure 1.1*** illustrates how accuracy of different data-driven models of various complexity for the word sense disambiguation task scale with the size of the annotated corpus they were trained on. All models show a steady gain in performance up to the data set 2000 times bigger than the original, with simpler model outperforming the rest on the smaller data set but falling behind as the size increases. Banko and Brill conclude these results with the following commentary: “While the observation that learning curves are not asymptoting even with orders of magnitude more training data than is currently used is very exciting, this result may have somewhat limited ramifications. Very few problems exist for which annotated data of this size is available for free” [9, p. 4].

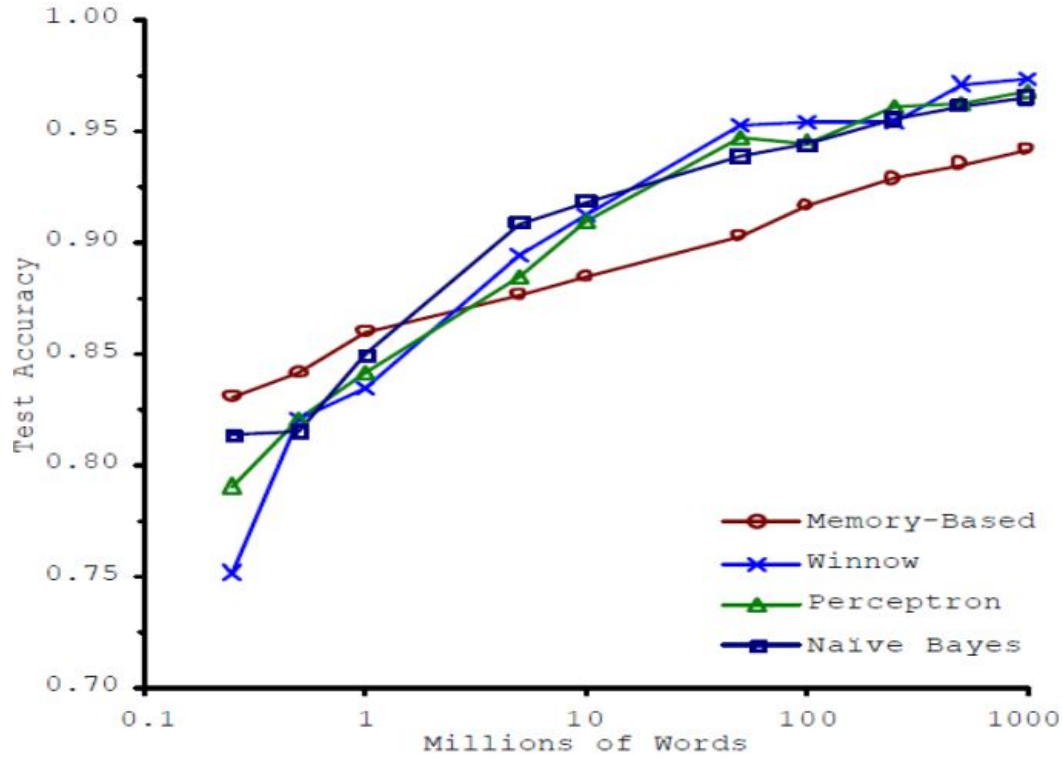

Figure 1.1: Dependence of Performance of various data-driven models for the natural language disambiguation task. Reprinted from [8].

1.4 Availability of data

Indeed, the training data availability still is among the hardest challenges for those who desire to try and implement the data-driven approach to more obscure and novel problems in Natural Language Processing which require unique special-purpose corpora. Sometimes these data could be obtained by elaborate filtering of large available annotated general purpose corpora like Penn Treebank[1], or other various corpora available on the Web[2], but even that option often is greatly limited for languages other than English for various reasons ranging from the obvious ones, like the lesser popularity of a language within the research community, to the very obscure and unexpected, such as copyright laws of the country the research group handling the corpus is situated. For example, the largest and the most advertised corpus of Russian - the Russian National Corpus[3] (over 300 million words) due to copyright reasons is only available online through the web query interface, which makes it useful only for the top-down theoretical modelling. The offline copy acquisition is only possible for a rather small part of the corpus (1 million words) and is associated with irritating legal rituals such as the signing of an actual physical copy of the license agreement, and rather pointless, given that this size is comparable to the size of the only alternative - the

[1] https://www.cis.upenn.edu/~treebank/
[2] http://www-nlp.stanford.edu/links/statnlp.html#Corpora
[3] http://ruscorpora.ru/en/index.html

OpenCorpora project[4], an open source community-driven annotated corpus of Contemporary Russian which is freely available for download.

For many NLP tasks though the filtering option is not feasible due to their specificity. The most obvious example here, again, comes from the Machine Translation, which requires substantial amounts of sentence-aligned, or at least paragraph-aligned, textual data for the implementation of a reasonably well performing data-driven system [8], and for many language pairs these data just do not exist anywhere. Same goes for the problem of automatic evaluation of the stylistic quality of texts with data-driven methods (which is the inspiration behind the practical part of this thesis). For the problem to be straightforward one requires no more than the large data set of short texts written by a large selection of adult native speakers and evaluated by another group of adult native speakers using some form of ordinal or numerical system of measurement (preferably with overlapping, to measure the interjudge agreement score). Obviously, a set like that does not exist, and even though the way of constructing it from scratch through the direct work with human interviewees is obvious, it would take a lot of organisational and financial resources to do so.

1.5 Web mining

Given these problems with the acquiring of textual data, it is to no surprise that linguists turned to the Web as a potential source as soon as its size surpassed the size of manually collected corpora. The Web is huge and consists largely of texts, after all. First research works of using the Web as a corpus quickly proved that even the most basic approach to handling it: direct queries through commercial search engines with the usage of reported numbers of hits as frequency estimations, provides results comparable to those gathered from traditional corpora of smaller size [10]. It is worth mentioning, that from the time of these first papers the Web grew quite a bit: in their paper of 2003 Mayer et al. reported the latest estimation of the Web size at 3 billions of unique pages approximately [11], and now in 2016 the size of the Web is estimated at 45 billions[5].

The Web-based approach to Corpus Linguistics quickly developed into rather autonomous discipline with its own structure, classifications, and methodology, to the point that some argue that the recognition of the *Web Linguistics* as an independent field should be considered [12], [13].

There are two main ways to use the Web as a corpus: direct online query to the Web through a search engine, and a Web corpus extraction for a following offline processing and continuous usage.

The former one has several very strong advantages: it is always ‘up-to-date’ with the language use on the Web; it always has the biggest possible size for a general-purpose web corpus; it does not require any storage space on the user side. However, the main downside of this approach is the reliance on commercial search engines such as Google, which obviously, are not intended for a usage in linguistic research and therefore lack many

[4] http://opencorpora.org/
[5] http://www.worldwidewebsize.com/

features crucial to a good corpus interface. That problem was predicted back in 2003 by Adam Kilgarriff, who proposed the creation of completely autonomous Linguistic Search engine for scientific purposes [14]. Since then there were several attempts to build one, but none were inspiring enough to be really accepted by the research community. Though, the most promising one - the WebCorp LSE[6] - is still in development, so there is hope yet.

The latter approach also has a range of strong and weak sides. Baroni and Ueyama [15] and Spousta [16] list following:

- Web mining gives us the ability to construct huge offline corpora with very little effort for a large selection of languages, including many of those without any easily accessible traditional corpora of comparable size.
- The representation of genres in mined corpora is generally more diverse than even in classic balanced corpora.
- Even if the goal is to create a small special-purpose corpus of very specific data, the vastness of the Web usually allows to do it easier and get the size larger than expected.
- Due to the insane diversity of web pages' internal structures, markup languages, metadata organisation and page encodings mined corpora tend to require a lot of very reckless post-extraction processing and usually are quite noisy even after that.
- Web is constantly changing, which in most cases makes a repetition of the mining process with exactly the same results impossible, and various legal and ethical reasons make the distribution of mined corpora a challenging endeavour; and together these two issues make scientific results achieved with the usage of Web corpora close to unreproducible.

1.6 Task-oriented data search

The advantages and disadvantages of the mining show a complex tradeoff of **development time**, **processing time**, **corpus quality** and **algorithm complexity** for the web mining, analysis of which is way beyond the scope of this thesis. This tradeoff is unavoidable when it comes to the construction of balanced web corpora. However, due to the anisotropic structure of the Web, it could be worked around in the creation of unbalanced special-purpose web corpora.

Linguistic phenomena tend to be unevenly distributed through the natural language texts of different purposes, genres or time periods, and if the goal is not to analyze the usage of the particular construction, but to collect as many naturally accuring examples of it as possible, it is better to decide on which texts are more likely to contain them, and which are of no use.

Thus, in construction of special-purpose web corpora it is not reasonable to start with the unsupervised web mining of random web-crawled textual data and post-processing it to extract desired fenomena occurrences, but first to think if there is a localized source on the Web, which might host texts with higher concentration of the data needed.

[6] http://wse1.webcorp.org.uk/home/index.html

For example, Web sites like Internet shops, movie, restaurant or hotel databases have a lot of *user reviews*, which are basically texts created with the intention to express the author's opinion on something, and often have a rating value and reviewer's ID attached within the metadata. This makes data mined from these sites priceless for Sentiment Analysis and, along with the Twitter social network and its excessive usage of emoticons, pretty much power up the whole field for the last couple of years (see [17], [18] and related works).

The main power of that approach to the task-oriented mining does not come from the higher speed of relevant data extraction, but from the higher quality of it. With limiting of the mining scope to a really small number of relevant sites it is possible to tune the processing directly according to the way textual data are stored on them.

1.7 Source-oriented task search

The significance of this advantage lead to that the most popular sources for the Web mining like Twitter, Wikipedia, IMDB developing their own stable tools for harvesting the data. Twitter allows the tweet data access through their own API[7], Wikipedia has a functional API for data extraction developed by researchers[8], and IMDB maintains the easy-access data mirrors[9].

The availability of data through these points of access leads to ever increasing interest to these particular data hubs in the research community, and with more and more inventive ways to study and apply these hoards of easily accessible textual information. The Twitter-based research in the last couple of years spanned way beyond Linguistics and Natural Language Processing scopes of interest, which lead to emergence of very exciting interdisciplinary studies. For example, tweets processed with NLP tools were used in attempts to predict stock market behaviour and electoral results ([19], [20] and related works).

1.8 Wikipedia as a localised and homogeneous source of data

Wikipedia is one of the oldest and biggest collections of multilingual textual data on the Internet. It was created 15 years ago and currently is ranked 7th most popular website in the world[10]. According to Wikipedia, Wikipedia has over 38 million articles in over 250 different languages[11].

Here I highlight the most valuable properties of Wikipedia from the point of view of Corpus and Computational linguistics and provide theoretical motivation on why I think these properties are undervalued and how can they be of use.

[7] https://dev.twitter.com/overview/api
[8] https://dkpro.github.io/dkpro-jwpl/
[9] http://www.imdb.com/interfaces
[10] http://www.alexa.com/siteinfo/wikipedia.org
[11] https://en.wikipedia.org/wiki/Wikipedia

1. **Size** - Wikipedia is very large, which, as discussed in 1.3, is important to modern data-driven methods based on Machine Learning and Bayesian Inference.
2. **Multilinguality** - Wikipedia is multilingual, with clear connections between articles about same entities in different languages, which is especially important for languages less covered by corpus research and has value for interlingual research parallelisation. Also, the structure of Wikipedia is the same for all languages, which makes all methods and applications without internal linguistic motivation easily adjustable between different language versions.
3. **Stability** - Wikipedia is one the most stable large textual data sources on the Internet. Due to the robust structure of article handling, the policy on collecting full edit history for all articles (with full texts, time of creation, and unique IDs for each version), and monthly dumping of all versions into backup archives, which are stored for about a year each, Wikipedia is (to my knowledge) the only web mining source for which it is possible to mine the data with the ability to repeat the same mining algorithm and get **exactly the same data.**
4. **Interconnectability** - Wikipedia articles refer to one another with semantically motivated and categorized connections which is priceless for research on Semantics and Linked Data. The value of this unique property of Wikipedia is already recognized and the DBpedia[12] project studies it for almost 10 years now.
5. **Revisions** - Full history of every version of every article stored indefinitely not only provides stability of source data, but also is valuable and unique data itself. Metadata attached to all revisions allow for elaborate data slicing and subset construction including, but not limited to, diachronic corpora, analysis of texts' evolution through time, and even personal language and competence modelling of registered editors.
6. **License** - Wikipedia texts and its dumps are licensed[13] under Creative Commons Attribution ShareAlike license[14] and GNU Free Documentation License[15], which ensures the freedom of data collection and allows researchers to freely share datasets and corpora extracted from Wikipedia.

Due to these qualities and availability Wikipedia does attract a lot of researchers from many fields[16] for many years, but it is still growing and developing, and still has a lot to offer.

1.9 Towards a corpus of edits for the text quality evaluation

In this thesis I focus on the revision history of Wikipedia as a potentially vast source of highly specific textual information - patterns for stylistic improvements in natural texts. I describe the creation of a special-purpose data extraction pipeline which parses the revision history of Russian Wikipedia and extracts all sentence-level user edits applied to the articles

[12] http://wiki.dbpedia.org/
[13] https://wikimediafoundation.org/wiki/Terms_of_Use
[14] http://creativecommons.org/licenses/by-sa/4.0/
[15] http://www.gnu.org/copyleft/fdl.html
[16] Google Scholar search with the keyword "Wikipedia" gives more than 20 thousands results for this (2016) year alone.

throughout their evolution. The corpus formed with all extracted edits is sampled, analyzed, and the means of building the branch of the corpus which would consist mostly of *stylistic edits* are discussed and tested.

The whole process of data manipulation is built with the intent to be as language-independent and modular as possible. In the perfect scenario the pipeline would depend only on language-specific training sets and function perfectly for another language if these sets are replaced.

If successful, the main field of application of a corpus of extracted stylistic improvements would likely to be the construction of automatic text quality evaluation systems.

Text quality evaluation is a rich research field with a lot of work done within it. In her comparative study of different text quality evaluation approaches Karen Schriver divides methods on text-focused, expert-focused and reader-focused and concludes that the reader-focused methods a the reader-focused are the most effective:

“When practical considerations such as time and expense allow, reader-focused methods are preferable to text-focused and expert-judgment-focused methods because they shift the primary job of representing the text’s problems from the writer or expert to the reader.” [21, p. 15].

Based on this notion, the dataset mistakes and problems in the text somehow highlighted by the end readers is the ideal source of information on the text quality, and this is exactly what the Wikipedia revision history provides. Thus, extraction of the user edits corpus from the revision history is required to check the following set of hypotheses (all assume the scope of the same language):

(1.1) User edits in the Wikipedia revision history do contain information on the properties of bad (requiring editing) Wikipedia articles.

(1.2) User edits could be automatically extracted from the Wikipedia revision history in a consistent manner in a reasonable amount of time to form a corpus big enough for modern data-driven algorithms.

(1.3) The information on stylistic faultiness of Wikipedia articles could be separated from the information of factual faultiness.

(1.4) The information on stylistic faultiness could be generalized beyond the extracted dataset and could be used to assess the stylistic quality of random Wikipedia articles in agreement with the human assessment of the same articles.

(1.5) The information on stylistic faultiness could be generalized beyond the scope of Wikipedia articles and used to assess the stylistic quality of random texts in the in agreement with the human assessment of the same texts.

In the hypotheses (1.3) - (1.5) I follow Bronner and Monz [22] in the assumption that user edits in the Wikipedia revision history could be separated in two different groups: ***factual edits***, and ***stylistic edits***. Examples (1.6) and (1.7), in which {} marks deleted text, <> marks inserted text, are factual and stylistic edits from Bronner and Monz [22].

(1.6) Over the course of the next {two years} <five months>, the unit completed a series of daring raids.

(1.7) In 1973, he {helped organize} <assisted in organizing> his first ever visit to the West.

Examples (1.8) and (1.9) are factual and stylistic edits from this study ({} - deleted text, <> - inserted text).

(1.8) V 1980 godu Abovina-Egidesa vyslali za granitsu, on žil v {Pariže} <Kreteje pod Parižem, gde i byl pohoronen>.
In 1980 Abowin-Jegides was deported, he lived in {Paris} <Créteil near Paris, and was buried there>.

(1.9) {Po} <Soglasno> analizam, {provedennym} <sdelannym> medikami bol'nitsy, gde umer Met'yu, on byl VIČ-položitelen na moment smerti.
{By} <according to> tests {performed by} <made by> physicians in the hospital where Matthew died, he was HIV-positive at the moment of death.

Initial design of this study included the texting of all listed hypotheses, but due to time limitations the testing of hypotheses (1.4) and (1.5) will not be covered in this thesis and will appear in the future work.

1.10 Target language of the study

Russian was chosen as the main language for this study due to following reasons:

- Part of the study requires manual classification of randomly extracted subset of edits - the task which should preferably be done by a native speaker. With Russian as a target language I was able to do the classification myself.
- As it was discussed earlier, Russian suffers from the lack of available corpora, which is why it is more important for Russian to explore new venues of textual data extraction for research purposes.
- Currently Russian Wikipedia is in the top 10 by size and in the top 5 by depth among all Wikipedias[17], which makes it one of the top choices to analyze how methods and results achieved in previous works for English Wikipedia hold if applied to another language.

[17] https://meta.wikimedia.org/wiki/List_of_Wikipedias

1.11 Thesis outline

The remainder of the thesis is structured as follows:
The beginning of Chapter 2 is dedicated to the analysis of practices of data usage in the modern data-driven Computational Linguistics research, its flaws are discussed and general benefits of the usage of Wikipedia as a data source, regardless of the task, are presented.

The chapter continues with detailed descriptions of Machine Learning methods used throughout the project, theory behind them and reasons why they were chosen over other possible approaches. It contains condensed descriptions of various parametric and nonparametric Machine Learning methods with links to practical part of the thesis and has an emphasis on Neural Network modeling and Dense Feature Representation which are the most important concepts for the project.

Chapter 2 ends with the part on Wikipedia mining. Previous research on the usage of Wikipedia revision history is presented. Important decisions in the design of the study, including deviations from previously commonly used methods are described and motivated.

Chapter 3 presents the Wikipedia data processing algorithm and the edits' corpus creation. All modules in the edits extraction pipeline, as well as interaction between them are described. The chapter contains detailed descriptions of creation, testing and evaluation of Sentence Splitter and Sentence Aligner extraction modules written in Python, which contain main implementation of the edit extraction procedure. The 'wrapper' program written in Java, which handles the extraction and dewikification of raw texts of Wikipedia revisions and controls the general flow of the extraction pipeline is also described here.

Chapter 4 is dedicated to the problem of refining the extracted corpus by throwing out edit pairs which do not represent stylistic editing. The task is reformulated as a binary classification problem between stylistic and factual edits, and different Machine Learning algorithms for this classification task are described and compared. Several approaches to the generation of training and test sets are discussed. Chapter concludes with the comparison of the best classifier's performance to the state of the art results of analogous systems in published papers.

Chapter 5 provides the final concluding discussion as well as the outline of the future work.

Chapter 2

Theory, data and method

2.1 Theoretical motivation: Why Wikipedia?

As it was mentioned in 1.8, Wikipedia has some unique properties which make a desirable source of data and an object of research for many fields including but not limited to the data-driven linguistic modelling. Following chapter is dedicated to the more detailed discussion on the reasons why in some cases Wikipedia might be more preferable as a data source compared to other methods of data gathering.

2.1.1 Hidden intrinsic disadvantages of the pure data-driven approach

According to Wallis [2], the integration of the the model-driven and the data-driven approaches formed a **unified theory** of a linguistic research work based on the cyclic approach presented on the **Figure 2.1.**

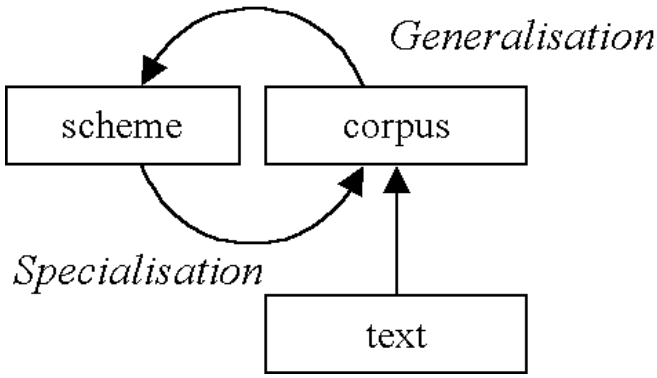

Figure 2.1: Wallis' scheme of a cyclic approach to the work with a corpus. Reprinted from [2].

Concept behind the scheme on the **Figure 2.1** explained in the following quote:

"A cyclic point of view accepts both (a) new observations generalise hypotheses or focus theory and (b) theory is needed to interpret and classify observations.
This loop is not a closed (hermeneutic) circle but an evolutionary cycle. Each passage around the loop enhances our knowledge by refining and testing our theories against real linguistic data. **A cycle can involve a single experimenter or a community of linguists debating results and evaluating hypotheses.**" [2, par. 2.2]

The highlighted part of the citation indicates, that the researcher in **Figure 2.1** is incorporated within the scheme block: scheme here is a part of the researcher's mind and

the cyclic process of scientific inquiry optimizes this model through generalisation and specialization. As the model and the researcher are inseparable, the learning of the model means the learning of the researcher.

Pure data-driven methods of text analysis and language modelling which incorporate Machine Learning (Figure 2.2), however, put the researcher outside of the optimization learning cycle. Here researcher interacts only with the scheme, which in that case is designed to autonomously interact with the corpus and proceed through the optimization cycles.

The learning of the model does not represent the direct learning of the researcher. Researcher can only observe and measure performance of the model on the desired task given the provided corpus. This does not mean that researcher does not learn anything. Researcher learns how to create the most efficient schemes which, given particular data, provide the best results for the particular linguistic task, but it is not the same as learning and understanding the language directly.

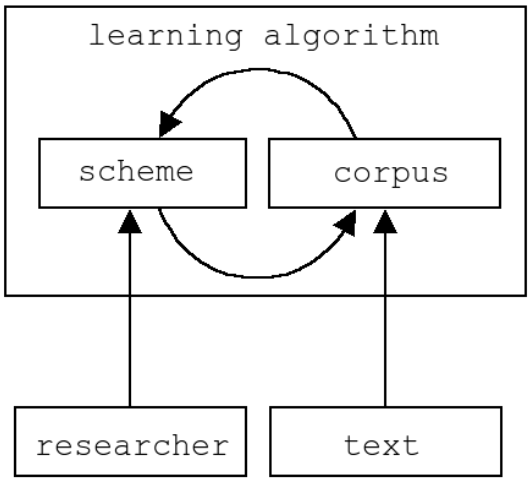

Figure 2.2: Scheme of a cyclic approach to the work with a corpus using data-driven machine learning. Modified picture from [2].

So, what is the difference between the direct learning and the learning by proxy? Is there a need to distinguish them in contemporary linguistics, given that the final applications of all models - either machine learned, or created by humans - nowadays are done through computational means?

To answer that question without drowning in details, it is reasonable to define **language**, **knowledge** and **linguistics** at the maximum level of abstraction. I don't intend to incorporate an epistemological debate into this paper, so all basic definitions will be declarative, and all derivations will be done according to the basic formal logic.

To define **knowledge** (K) I will use the most basic definition by Descartes [25]:

(2.1) Knowledge exists, because a knowing person exists. $\exists$ K**.**

Therefore, the **language** (L), following the *functional definition* [8] is:

(2.2) L = {V , F: V $\rightarrow$ K}, V - **language vocabulary** (set of all possible finite lines), F - **function**, transforming elements of V into K[18].

Therefore

(2.3) $\exists$ $K_v \subset K : K_v = F(V)$

Let's call K_v **Verbal Knowledge**.

Consequently,

(2.4) $\exists$ $F^{-1}: K_v \rightarrow V$, $\forall$ $k \in K_v : F(F^{-1}(k)) \approx k$.

Meaning, that F is invertible, and transformation of an element of **Verbal Knowledge** to a string and back creates an element of Verbal Knowledge almost equal to the original.

In the given axiomatics these definitions are applicable for any language. Now, let's consider a **natural language** N. The following rule is the key to this abstraction:

(2.5) $\forall$ $N = \{V_n , F_n : V_n \rightarrow K\}, F_n \in K \backslash K_v$

Which means that the function transforming a natural language text to a **Verbal Knowledge** is within the scope of **Knowledge** itself, but is not within the scope of **Verbal Knowledge.** Let's call this complement a **Nonverbal Knowledge** and define it.

(2.6) $K_{nv} \in K , K_{nv} \cup K_v = K , K_{nv} \cap K_v = \varnothing$

The motivation behind that is as follows:

(2.7) Any person fluent in some natural language N **knows** F_n, therefore $F_n \in K_p \subset K$

(2.8) If $F_n \in K_v$, then, given (2.7), any person fluent in some natural language would be able to generate F_n^{-1} (F_n) - a complete description of the rules of text-to-knowledge transformations of this language using the language.

There are people fluent in a natural language who can't present a formal description of this language, therefore (2.8) is not true. Therefore, (2.5).

Now we can define the **goal of linguistics** for any natural language as a search of a function $F_{vn} \in K_v$ which is a verbally describable approximation of F_n.

[18] Here we assume fixed mental and environmental contexts for simplicity

(2.9) $F_{vn} \in K_v, F_n \in K_{nv}, \forall v \in V_n : F_{vn}(v) \approx F_n(v)$

With this abstraction we can see, that the scheme on a Figure 2.1 operates within the scope of **Verbal Knowledge** K_v. Therefore, a linguistic research based on that scheme advances towards the declared goal (2.9). Research scheme on the Figure 2.2, however, does not necessarily operate within the scope of **Verbal Knowledge** K_v, as the learning cycle does not include humans and therefore skips the verbalisation step, which would keep it within K_v. Thus, it can not be considered a linguistic research, as the knowledge about the language created in the process is not verbalized.

This, however, does not mean, that the pure data-driven approach to linguistics based on machine learning is a strictly applied discipline and the empirical knowledge obtained with it provides no theoretical value apart from the self-oriented knowledge on how to build more efficient data-driven systems for Natural Language Processing.

Systems like that are built to extract the knowledge directly from texts - they do create a function G_n which is updated through a learning algorithm to improve as an approximation of the desired F_n, but they do not stay within K_v (2.3). However:

(2.10) $F_{vn} \in K_v, G_{vn} \in K_v, F_{vn} \rightarrow F_n, G_{vn} \rightarrow G_n, G_n \rightarrow F_n \Rightarrow G_{vn} \rightarrow F_n, F_{vn} \sim G_{vn}$

Which means, that if artificial machine learning system approximates the real natural language well enough (its results are not worse than human performance on the same task), then the artificial approximation can be studied instead the language itself and the results of this study **will satisfy the goal of linguistics within the defined abstraction** described in (2.9).

Researchers usually have more control over models created by means of machine learning than the real-world alternatives, which are human annotators or interviewees. Models can be freely retrained on different controlled sets of data, can be localized to approximate only one particular aspect of a natural language, can be sampled and resampled endlessly, and you don't even have to pay them.

However, the research on a trained model should be held up to the same standards as a research done using the data from human participants. The machine learning approximation of natural language is the equivalently reliable source to the real language data only in the ideal case, and as there are no ideal models just yet, therefore sanity checks against real data should be regularly conducted.

To summarize, a research oriented on the creation and improvement of data-driven models for solving various Natural Language Processing tasks does not directly contribute to the theoretical linguistics, as it does not improve the human understanding of natural languages. The knowledge obtainable through usage and study of these models, however, does hold theoretical relevance, but only if created models' performance is comparable to humans'.

In (1.3) it is stated, that the performance of a data-driven model heavily depends on the size of data it is trained on. This highlights the first answer to the “Why choose Wikipedia over other sources of textual data for the data-driven research?” question - it is size, homogeneity and accessibility of Wikipedia.

Wikipedia contains large quantities of textual data (1.8) which are easily accessible to anyone from anywhere. The data is stored in the same unified and clearly defined format, which allows researchers to quickly construct efficient extraction programs, and even if some data is lost, it could be easily restored by repeating the same extraction algorithm to the same instance of Wikipedia data, which are stored indefinitely.

2.1.2 Reproducibility, Legality, Ethicality, and Circularity within the data-driven linguistic research

The stability of Wikipedia not only allows to sometimes conveniently restore some lost data but also helps with another, more systemic problem with data-driven linguistic research.

Data-driven models defined by the training data. Feature sets, learning algorithms and their parameters are tuned for the extraction of the most generalizable information from the particular data set they are trained on. The change of data changes the model, therefore an extensive theoretical research preferably should be performed on the same data set.

This notion highlights the first problem of the data-driven linguistic research, which were briefly mentioned in the Introduction - the problem of reproducibility of scientific results. To repeat or continue someone’s data-driven study a researcher should ideally have the exact training data that was used in the original study, at least as a starting point, and the modification to the set should be introduced in the controlled manner, otherwise the continuity will be compromised.

This is especially important for Neural models (more in 2.2.2) due to the fact, that despite their popularity and effectiveness, we are still far away from understanding how exactly they perceive the data, and therefore repercussions of training set changing for the Neural model are absolutely unpredictable (more on the topic). There is a rather famous picture (Figure 2.3) from Szegedy, Christian, et al. [26] which gives some insight on how unpredictably sensitive neural models can be. Here the left column shows pictures correctly recognized by the image classification neural model, the right column shows pictures incorrectly classified as *an ostrich,* and the middle column shows the difference between left and right pictures.

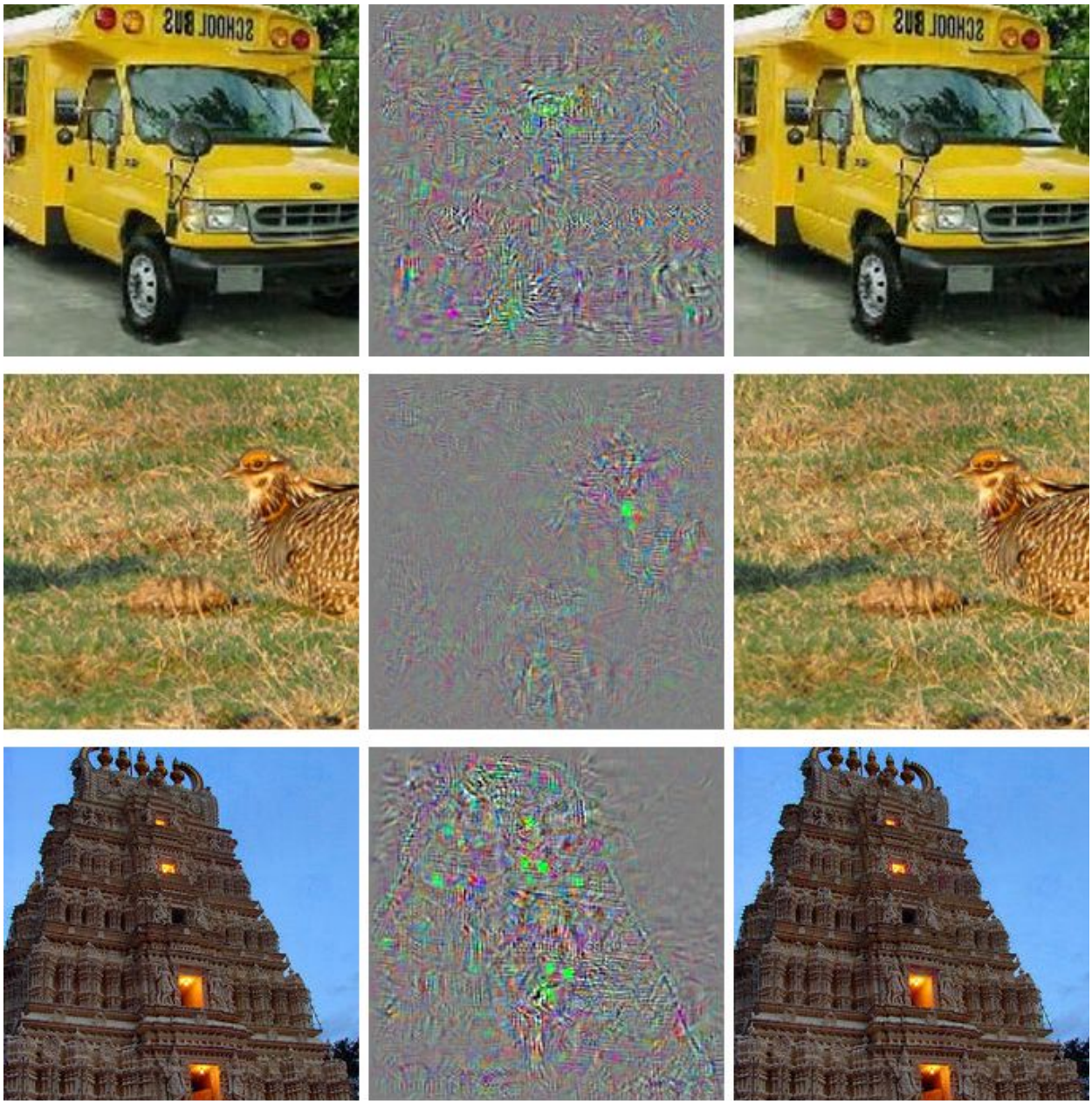

Figure 2.3: Minimal distortion required for misclassification of images by the AlexNet model. Reprinted from [26].

Still, works that make use of someone else's data are quite rare and the interconnectivity within the field consists mostly of copying or deriving new model designs and algorithms from the previous research, disregarding the data. This is due to the fact that the most corpora and datasets used in the modern data-driven research are extremely large, and almost all extremely-large corpora are Web-based.

According to Baroni and Ueyama [15] and Spousta [16] the problem of Web-corpora sharing is mostly legal - web crawling is impossible to direct and it is a lot of work to assess the copyright of all texts extracted from more or less random web pages to ensure that providing free access to texts within the corpus does not violate someone’s rights of ownership. Also, due to the fact that the Web is constantly changing, it is useless to share the parameters and

the algorithm of the Web-based corpus construction, as the repeated crawl will never create the same corpus.

Stability and free license of Wikipedia (1.8) make Wikipedia-based corpora and datasets unaffected by the aforementioned problems of most web-based corpora. Corpora extracted from Wikipedia can be shared freely, and publication of the list of revision IDs involved in the mining makes it possible to repeat it exactly.

Another problem of the distribution of web corpora is raised by Adrien Barbaresi in [14]:

"If data are supposed to be in the public domain, they can be considered free of copyright concerns, but should not be treated as unworthy of questioning on an ethical level. For instance, possible privacy issues can arise from intersections that are made in a large set, which, according to the metadata delivered with the corpus, can lead to identification of individuals, their localization, or revelation of patterns in their daily lives." [14, pp. 63-64]

This ethical problem is especially significant for Social Networks, and they are responsible for a significant part of texts on the Internet. For example, Facebook and Twitter both currently are in the top10 of the biggest sites in the world[19].

Barbaresi gives some examples of methods like *masking* and *scrambling* which could be used to hide any possible personal information, but concludes, that these methods are ultimately very invasive and seriously devalue processed corpora for research purposes [14].

This ethical problem is also nonexistent for Wikipedia-based corpora by design - contributions to Wikipedia are intended for large audiences and it is safe to assume that most contributors are aware of that. There is a possibility of intentional personal information reveals in *vandalistic edits*, but this requires further research and they are account for rather insignificant part of the data (~3.5% of manually classified set of edit pairs, including both vandalism and correcting rollbacks).

The last problem of traditional and web corpora used in data-driven CL research discussed in this thesis is the problem of *circularity.* In [27] Riezler describes it as "A problem which arises in the application of machine learning methods to natural language data under the assumption that input–output pairs are given and do not have to be questioned" [27, p. 2].

It is stated, that in most cases training set preparations are done by experts who operate within the same theoretical framework for both feature extraction and data labeling, thus the theory is tested not against real data, but against the data already partially interpreted by the very same theory, creating the self-confirming circulation of baseless knowledge within the model [27].

As one possible solution to the problem of circulation, Riezler proposes the search of weakly-labeled data "in the wild": on the community-driven crowdsourced internet resources

[19] http://www.alexa.com/topsites

like IMDb, Wikipedia, Yahoo Answers and so on, because these data are created by the large variety of non-experts and the labeling process is not the sole intent but is “naturally occurring” during other people’s activities [27].

This conclusion is closely related to ideas behind the practical study on user edit extraction presented in this thesis and provides further evidence in support of the theoretical hypothesis on unprecedented value of the Wikipedia as a data source for the data-driven linguistic research.

2.1.3 Concerns about the balance

Balance is one of the main concerns of traditional Corpus linguists when it comes to Web corpora and corpora constructed of texts of one genre [28]. Obviously, Wikipedia is both, so Wikipedia-based corpora are likely to be unbalanced.

However, a special-purpose corpora are unbalanced by definition, so the problem may arise only for the general-purpose corpora extracted from the Wikipedia. I argue, that as a freely accessible and stable benchmark data for a data-driven research, which would be used mainly for the data interconnectivity of research papers, the unbalanced corpus is good enough. More so, it could be beneficial for the interlingual research, as the style of Wikipedia articles is more or less the same across languages.

Also, only seasoned Wikipedia contributors create new articles in the wiki-style from the revision one. Research on frequencies in Wikipedia contributions showed, that it follows Lotka’s law with the power value of 1.65 for the tail of infrequent contributors [29]. This indicates a reasonable probability, that there are enough articles with non-wikified versions within the revision history to create a corpus unaffected by the general style of Wikipedia articles.

Still, separate experiments designed to determine the level of generalisation of results achieved using Wikipedia data to different language domains are needed before setting the final judgement on if the knowledge generated through the study of the language of Wikipedia is applicable to the general language or not.

2.2 Methodology

The process of corpus construction from Wikipedia data presented in this project relies heavily on various Machine Learning algorithms and techniques for Natural Language Processing, some of which are used in traditional ways and some are changed to fit specific challenges of the project. This section is dedicated to the presentation of the field of Machine learning and its connection to Natural Language Processing, overview of methods used in the project, and motivations behind the choice of these methods.

2.2.1 Machine Learning

The general motivation behind the application of Machine Learning to various NLP tasks derives from the functional abstraction of natural languages presented in in 2.1 and is mostly practical in its nature.

2.2.1.1 Functional abstraction of language

Rule-based systems for NLP are derived from various linguistic models designed to describe languages as separate existing entities with complex internal structures, in other words - designed to create language understanding. This dependence on complex models burdens rule-based systems immensely, because no matter how simple the task is - the rule-based model either has to work with it within the context of the whole language model, or has to be separated from the model with the set of additional restrictive and generalizing rules crafted by the researcher, which often takes a lot of work and the result is excessive in its computational strategy[20]. Rule-based systems by design have to create some human-like level of understanding of the text from the input and only after that they are able to create the required output.

Functional abstraction, on the other hand, allows us to ignore the language understanding part and work with the language use directly. Kyunghyun Cho describes the functional approach to language modeling the following way:

"This function (called language) takes as input the state of the surrounding world, the speaker's speech, either written, spoken or signed and the listener's mental state Inside the function, the listener's mental state is updated to incorporate the new idea from the speaker's speech. The function then returns a response by the listener (which may include "no response" as well) and a set of non-verbal action sequences" [8, p. 9]

If language is treated as a complex function which links a set of textual and contextual inputs into another set within the same space, then it is possible to treat it as a mathematical function which could be decomposed, projected, and approximated within some local subspace of the particular language-related task.

Thus, within this abstraction for the creation of a system for automatic execution of some language-related task it is possible to gather the data for this task from the real world, organize it in a set of *input-output*[21] pairs, treating the outputs as results of application of the same unknown function to corresponding inputs, and approximate this function using various mathematical methods.

[20] For example, even if the task only requires to extract all Adjectives from a raw text, a rule-based system would have to perform full POS-tagging of the text and only after that it would be able to extract Adjectives and discard the rest

[21] Input can be empty - in this situation we assume that the outputs are results of the application of some function to unknown input

In the context of language use, learning is the gradual approximation of an ideal intersubjective language model from a chain of (linguistic) action-response events. Within this setting Machine Learning methods construct a similar process, because *the gradual automatic approximation (learning) of unknown functions from data generated by these functions* is the core goal of Machine Learning [8].

This goal is closely related to the problem of mathematical optimisation, though there are some subtle differences (for more formal discussion see [37]). Most Machine Learning problems are solved through the reduction of the approximation task to the optimisation problem, either making a hard assumption about the form of the true function (parametric algorithms) or trying to exploit the available (training) data as much as possible in the approximation task without any limiting assumptions about its form (nonparametric methods).

2.2.1.2 General properties and structure of Machine Learning algorithms

Limiting assumption about the form of the true function in parametric methods makes the computation faster, as it restricts the class of possible best approximations, and the presence of fixed number of explicit parameters makes it easier to tune models and interpret their results. Nonparametric methods generally fit training data better, but often suffer from overfitting, trusting the training set (with all its possible outliers) too much, which leads to the need in additional methods of fighting with overfitting of a single model such as bagging and boosting. Additionally, nonparametric models are much slower to train due to (ironically) a very large number of implicit parameters which grow in numbers with the size of training data are had to be calculated. Middle ground methods usually classified as Deep Learning algorithms (more in 2.2.2), which gained a lot of attention over past years, are trying to take the best of both worlds. They are technically parametric as they do make a limiting assumption on the form of the approximated true function, but the expressive power of the class of functions which this parametric model can generate is so large, that in practice it is almost indistinguishable from nonparametric models, and yet they retain most benefits of parametric models regarding the ability to distinguish and ignore outliers in training data.

Most parametric Machine Learning algorithms can be described with a standard procedure which can be summarized as follows:

1. Input data (if defined) are standardised in structure, so for every data point there is the same amount of numeric values (*features)* describing it in some way. Each feature should have values of the same type for each data point.
2. Output data are standardised following the same procedure.
3. For the given data a Machine Learning model defined by the finite set of parameters is constructed in a way, that given an input vector of the same structure as a feature vector from the input data it generates the output vector of the same structure as in the output data.
4. Model parameters are initialized in some way (initialisation could be random, even though the best results are usually achieved with more advanced initialisation methods)

5. A special *cost function* is constructed from the model parameters. The *cost function* indicates how correct is the current state of the model in predicting known outputs given known inputs, in other words, it signifies the *risk* of incorrect prediction of the output by the model [37]. There are several commonly used types of cost function, all of which are designed to indicate some kind of difference between the predictions and the real output values. These functions include various mean errors, cross-entropy, cosine proximity etc. The correct choice of the cost function is usually defined by the type of the task the model is trying to solve.
6. Initial value of the cost function is calculated.
7. Using a mathematical optimization algorithm, the optimisation step for the cost function is calculated, and all parameters of the model are updated accordingly. As with cost functions, there are many various commonly used optimisation algorithms and the optimal choice is not always clear. Optimisation algorithms are usually chosen either based on a number of vaguely defined heuristics based on the model architecture, structure of the data, type of the cost function etc., or through the trial and error, or with the combination of both.
8. Updated value of the cost function is calculated and compared to the previous one.
9. Steps 7 and 8 are repeated until the optimal solution is found, or another predefined stop criterion is achieved.

This is the general procedure for tasks with present input data, which are usually classified as Supervised Learning. Supervised Learning tasks are classified based on the structure of output data: prediction of categorical output is called Classification and prediction of numeric output is called Regression. The classification task is playing major role in the project, used classification models are described in 3.4.1 and Chapter 4.

The described procedure changes slightly in the case of Unsupervised Learning where the input data are absent, as in that case we can't actually use input vectors to calculate predictions, so the cost function has to be built based not on the real samples of the input data but on the assumptions about the piecewise-defined structure of the generating function.

The most common tasks for Unsupervised Learning include Clustering (identifying the most likely piecewise-defined structure of the generating function), Anomaly Detection (identifying how likely it is that the given output data-point is generated by the same function as all other given output data points), and similarity-based recommendation (identifying which one of the given data points is the most similar to the query data point, used in the project and described in 3.4.2). Some unsupervised methods (usually called Representation Learning) include automatic construction of supervised sub-tasks based on the internal structure of the analyzed data with the goal to extract latent information about the structure of the data used as an input for the auxiliary supervised task (a method from this category is used in the project and discussed in detail in 3.4.1.4).

There are many approaches to both Supervised and Unsupervised Learning (and to other classes of tasks which are not discussed here), so this brief overview will cover only algorithms and methods directly relevant to this project (either used, or considered but

discarded because of presented reasons) or to methods used in previous works (discussed in 2.3.1).

2.2.1.3 Memory-based algorithms

Memory-based algorithms are nonparametric supervised Machine Learning algorithms which do not incorporate any training and the learn by simply storing all the input-output pairs of the training data set in the memory. Assumptions about values on new data are based on a weighted combination of output values of training data points stored in the memory.

The most common Machine Learning algorithm of this class in K-Nearest Neighbours (KNN) algorithm, which predicts the value of a query data point by a weighted average of output values of k 'nearest neighbours' in the memory, which ate data points with input values closest to the query point by the distance metric used in the particular implementation of the algorithm. Distance metrics are usually chosen based on the structure on input vectors. The most simple variation of the algorithm (1-Nearest Neighbor) just returns the output value of the most similar data point stored in its memory.

Memory-based algorithms were repeatedly reported as strong competitors to much more complex Machine Learning algorithms on a number of common language-related classification tasks, despite being simpler easier to implement. However, their prediction quality heavily depends on the amounts of available training data and the algorithm itself relies on the assumption that the training data points spread almost equally over all possible input space (for all possible query points to have at least several neighboring points in the training set) and the performance drops significantly if this assumption is not met. Also (as with all nonparametric algorithms) with the increase of the training set in size the algorithm requires either a lot of computational power even with algorithm alternations such as K-dimensional tree search or Locality Sensitive Hashing which are more efficient than brute force pairwise distance computations.

Due to relatively small size of extremely high-dimensional training data in the main edit classification task in this project the question about the equality of distribution of the training data over the input space could not have been answered reliably, so it was decided against the Memory-based approach for that task. A KNN model is however used in the sentence alignment task within the edit extraction pipeline (3.4.2).

2.2.1.4 Decision Trees

Decision Trees belong to a class of nonparametric supervised classification algorithms which refer to the values stored in the training set for predictions on new data. However, instead of predicting the output label for a query data point based on its limited neighbourhood within the training set, as it is done in KNN algorithms, here the prediction is based in the structure of the whole training set. In this algorithm, prior to the classification of

new data, a binary decision tree for the available training data set is constructed following a simple top-down recursive procedure:

1. All training data features are transformed to the binary format[22]
2. Information gain of all binary features for the training data is calculated and the **best splitting feature** is determined.
3. Branches are formed from the current node, splitting the set into 2 subsets based on the value of the best splitting feature
4. 2-3 are repeated for both branches until the predetermined stop criterion (usually the maximal allowed depth of the tree or the minimal allowed size of a non-leaf node) is achieved, or until all data points in the branch share the same output label. Each node in which the tree generation was stopped becomes a leaf of the binary decision tree and gets an output label based on the majority vote within training data points in that leaf.

After the construction of the tree, for the classification of a new data point its input vector is simply run through the constructed tree and gets the output label of the leaf node it stops in.

Obviously, without any stopping criteria the tree construction algorithm would run until it fits the training data perfectly, which is rarely desirable, as the goal is to approximate the true generating function behind the data and training data is almost never represents it perfectly, so the perfect fit to training data always performs worse on new unseen data. However, it is not clear which stopping criteria would fit the particular task the best, so either repeated trainings with different stopping criteria is required (with following decision about the best model based on performances on the validation set), or an ensembling technique has to be implemented. Currently, the latter approach is used more frequently due to better and more reliable results.

The method of decision tree ensembling (often called Forest method) is derived from the statistical resampling method of *bootstrap aggregating* (or *bagging*) which was introduced by Breiman in 1994 [38]. The idea behind this method is to discard the repeated training of the same classifier on the same training set in order to deduct perfect model parameters and instead create N different training sets of the same size by repeated sampling with replacement from the available data and train N different classifiers with the same structure and parameters on these sets. The final decision is then created by the ensemble of these N decision trees by means of plurality voting.

The Random Forest algorithm [39] takes the described procedure one step further and applies bagging to both training data and features, so each tree is trained with the random uniformly sampled subset of features of the original feature set. This method decorrelates trees in the forest in cases of data with several strong features which are almost always chosen as the best splitting feature and improves the overall performance of the classifier.

[22] Meaning that, for example, a feature with 3 possible outputs ‘color’: {‘red’, ‘green’, ‘blue’} should be transformed into 3 features: ‘red’: {‘yes’, ‘no’}, ‘green’: {‘yes’, ‘no’}, ‘blue’: {‘yes’, ‘no’}

Random Forests are regularly reported among the best performing classifiers on various tasks, and it was the best performing classifier in the paper on the English Wikipedia edit classification by Bronner and Monz [22], which is the main reference for previous research in this project. The Random Forest classifier was therefore considered for the edit classification task in the present system. However, at the analysis step it was discarded due to differences in the feature structure used here and the one used in the reference project, as will be explained now.

It was proven that the learning time of Decision Trees is exponentially dependent on the height of the tree not only in the worst case, but on average [40], and the optimal height of a decision tree is dependent on the number of features in the training data. Bronner and Monz used a fairly low number of linguistically motivated features such as counts of inserted/deleted POS-tags (per tag), counts of inserted/deleted acronyms, named entities etc. This led to a reasonably sized model with sparse-represented features all of which are easily interpretable, which are perfect conditions for a Decision Tree-based algorithm. In the present project, however, dense-represented multidimensional features which were learned through unsupervised embedding algorithms from raw language data are used for the edit classification, which leads to models with hundreds of features uninterpretable by humans. The resulting large feature set means that the Random Forest algorithm would be too computationally dependent on the training set size for experimentation on a desktop machine. Whereas training was fast and provided good results on the small available manually classified training set, its computational complexity would explode if some automatic methods of reliable training set extraction were discovered and the size of training set was scaled up to tens of thousands.

Experiments with Random Forests with good results were performed nonetheless (as will be presented in Chapter 4).

2.2.1.5 Logistic Regression

Logistic Regression is a linear parametric binary classification Machine Learning algorithm. The core limiting assumption for this algorithm is that it assumes the true form of the decision boundary between two classes to be a straight line (N-1 dimensional hyperplane in N dimensional space) which makes it the most simple parametric classification algorithm. This algorithm operates according to the procedure described earlier in this section.

After the initialisation of the decision boundary vector which defines the decision boundary hyperplane in the feature space for given data, for each data point the algorithm projects it onto the boundary vector to get *score* values $\in (-\infty, +\infty)$ and applies *logistic function* $1/(1+e^{-x})$ to scores to get the probability estimations $p_n \in (0, 1)$ for each point. This estimates the probability of the data point being labeled as the default class by the model given the current decision boundary. Loss function is calculated as log-loss $L = -\frac{1}{N}\sum_{n=1}^{N}[y_n log(p_n) + (1-y_n) log(1 - p_n)]$ between predicted probabilities p_n and real labels y_n for the training data. This loss function extremely punished the model for high confidence

in wrong answers, forcing the model not only to try and get as many labels right as possible, but also to get unavoidable wrong answers as close to the decision boundary as possible, which indicates uncertainty in the label.

Having a very strong limiting assumption on the form of the boundary function, Logistic Regression is very resistant to the overfitting problem and due to its simplicity it is very fast to train and to use for predictions. While providing these benefits, the strong bias towards linear decision boundary also makes Logistic Regression model to underperform on more complex tasks, which means it is rarely the final choice in cases where the model performance is important. In Bronner and Monz [22] the Logistic Regression showed the worst results out of all tested models, scoring almost 2% less in the final reported cross-validation accuracy than the best model.

Nonetheless, Logistic Regression modeling is really useful in initial stages of practical Machine Learning model development as a data exploration tool. Applying logistic regression to the constructed feature set before trying out more complex algorithms does not take too much time and effort, as the model is very fast and simple, and provides a very valuable insight on how linearly separable training data points of two classes of the binary classification problem are in the constructed feature space. The performance of Logistic Regression classifier simultaneously gives information on the quality of chosen feature space if compared to the hard baseline of a majority classifier[23] and provides a soft baseline to more complex methods, setting the lowest boundary of performance they have to achieve and providing the reference point for measuring the *performance gain / execution time increase* ratio. In this project Logistic Regression algorithm is used in this way, which is described in 4.3.3.

2.2.2 Deep Learning

All Machine Learning algorithms described in the previous section have their strong sides and all are regularly used in practice. However, in the past couple of years all of them were constantly falling behind in terms of sheer state-of-the-art performance on the main task (regression, classification etc), being outclassed by various Neural Network based methods. Commenting on the matter Yoav Goldberg writes:

“The non-linearity of the network, as well as the ability to easily integrate pre-trained word embeddings, often lead to superior classification accuracy. A series of works (...) managed to obtain improved syntactic parsing results by simply replacing the linear model of a parser with a fully connected feedforward network. Straight-forward applications of a feedforward network as a classifier replacement (usually coupled with the use of pre-trained word vectors) provide benefits also for CCG supertagging (...), dialog state tracking (...), pre-ordering for statistical machine translation (...) and language modeling (...).” [41, p. 3]

[23] A classifier which always predicts the most common class in the training set, thus achieving C/N training accuracy, where C is the size of the most common class anN is the size of the training set

Following the trend, the main experiments in this work were done using Neural Network based Machine Learning algorithms, as a good performance on the edit classification task is crucial for the intended structure of the final corpus, and the concept of using pre-trained word and character embeddings as features lined up well with the concept of maximal algorithmic language independency of the extraction pipeline (more in 2.3.6).

2.2.2.1 Neural Networks

Basic Neural Network algorithms were introduced in 1950s [30] as brain-inspired computational mechanisms which consist of collections of simple computational units (*neurons*) organized in predefined structures which define the flow of information through the network. Each neuron takes finite number of scalar inputs and produces single scalar output which can be directed to any number of following nodes. Processing of inputs is done by applying weights stored in the neuron to inputs, followed by pooling (usually summation) of weighted results to get a single value and applying a predefined nonlinear function to this value to generate the final output of the neuron.

Usually neurons are organized in *layers:* all neurons in the same layer are characterized by the same[24] output. Layers serve the purpose of structurizing the model and control the flow of information through it. The first layer of a Neural Network consists of the chosen vector representation of the input units and the last layer generates vectorised output of predefined length. General structure of Neural Network is presented on Figure 2.4.

[24] It is not strictly forbidden to put nodes with different inputs in the same layer of a neural network and there were experiments with networks which had interlayer connections. This does not compromise the model and does not necessarily make it worse, but it makes the model harder to understand and analyze and generally avoided. Think of GOTO transitions in programming as analogue.

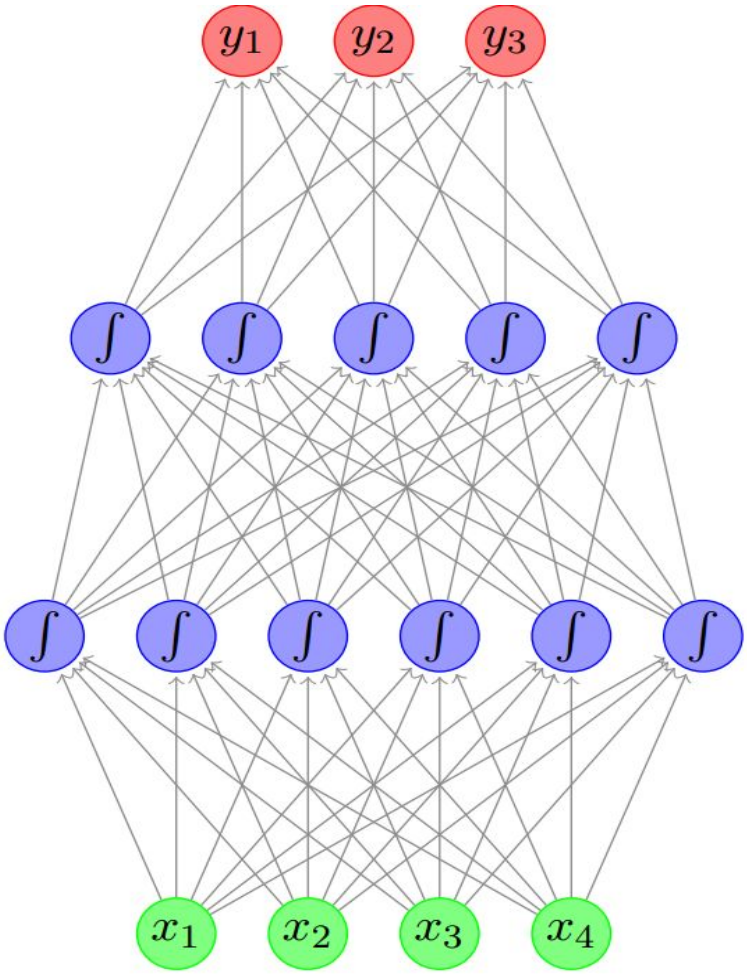

Figure 2.4: Simple Feedforward Neural Network with input layer (green), output layer (red), and two hidden layers (purple). Reprinted from [41].

In this network all nodes in a hidden layer take inputs from all nodes of previous layer and send their output values to all nodes in the following layer. Neural Networks of this structure are called Fully-connected (or Dense) Feedforward Neural Networks. After of the introduction of networks of this structure it was mathematically proven, that Feedforward Multilayered Neural Networks are universal approximators and *"Single hidden layer ΣΠ feedforward networks can approximate any measurable function arbitrarily well regardless of the activation function Ψ, the dimension of the input space r, and the input space environment μ"* [42].

Technically this means that additional layers are unnecessary, as everything can be approximated with one hidden layer. The problem of finding the Neural Network which approximates the given training set is NP-complete even in the most simple cases [43], so most of the times we have to settle for Networks which approximate *well enough*. Also, lately it was proven, that the *expressive power* of Neural Networks grow exponentially with depth, which allows multilayered networks with the same overall number of hidden nodes to approximate much more complex functions than single layered ones [44]. This means that with deeper structure it is easier to get the desirable performance through automated training

within the single architecture, while in single-layered design it will take a lot of different runs with different numbers of nodes in the layer.

This property of 'unreasonable effectiveness' of multilayered Neural Networks lies behind the emergence of the Deep Learning subfield within Machine Learning which specifically concentrates on the research of these multilayered models. Research within Deep Learning is not limited to Multilayered Feedforward Networks described earlier, but also involves creation of more complex network designs better suited for specific domains, such as Convolutional Networks for space-embedded structured information (especially 2-dimensional images) and Recurrent Networks (RNNs) for inputs of arbitrary length. The latter architecture is the most important one for NLP, as natural languages mostly consist of "units of varying length" of same types such as words and sentences, and the strict restriction on the unified form of input data in all other Machine Learning algorithms was always severely limiting, forcing researchers to use approaches akin to "Bag of Words" (described in 2.2.3) which completely disregard internal structures of sentences.

2.2.2.2 Recurrent Neural Networks

RNNs process inputs of arbitrary length by recursively calling the same fixed-length layer for the each element of the input sequence. This means that at the each recursive step the current RNN layer depends on the input vector of the current sequence element and the resulting vector of the previous layer[25]. This allows the important information to accumulate and flow through the sequence, reaching the end in a shape of a same-sized vector for any sequence [41]. The problem with the simple RNN approach is that it forces all information to go through the main stream of the processing network and be processed through linear combinations and nonlinearities at each step. This leads to information attrition in longer sequences and makes it harder for simple RNNs to pick up long-range dependencies if they are present in these sequences [45], and these connections are very common in textual input.

To offset the problem with long-range dependencies in texts for RNN processing the extension of RNN architecture was proposed. Long Short-Term Memory (LSTM) networks [46] operate as RNNs in general, but they also incorporate an explicit *memory state*[26] which is involved as an input at each step of the recursion and also can be updated at each step, but the access to the update is controlled by two trainable gates, represented with single-layer Neural Networks with sigmoid nonlinearity, which produce vectors of values in (0,1) range which is applied to the modified vector by pointwise multiplication. On each step of the recursion gates represent **'forget'** (how much of remembered information should be forgotten before the next iteration) and **'remember'** (how much of the information gathered from new input at this iteration should be added to the memory) transformations of the memory state. Gates learn with the network through the general backpropagation training procedure. In cases where input sequences exhibit consistent long-term dependencies the gate learns to preserve relevant information in the memory state without unnecessary

[25] At the first step of recursion the "results of previous step" vector is usually initialized with zeros
[26] Sometimes it is also called a *cell state*

updates which makes it easier for the network to incorporate these dependencies in the training process.

In the edit classification part of this project the basic state of the classified data is a pair of Russian sentences which represent two versions of the same sentence from two consecutive revisions of the same Wikipedia article (more in 2.3.5). These sentences are of arbitrary length and there are no preprocessing restrictions on positions of edits within the sentence scope. This makes LSTMs a compelling tool for experiments which not only focus on things that actually changed through editing but try to incorporate ***context*** of these editing transformations into the analysis. In section 4.3.5 various experimental designs for LSTM modeling are analyzed, and results are presented and discussed.

2.2.3 Feature Representation

The final important concept that needs to be covered in this section is feature representation methods for textual data. As Machine Learning for NLP operates within the functional abstraction, treating language as a mathematical function, textual language data is also has to be projected within numerical domain to make all usual mathematical methods and operations applicable.

2.2.3.1 Bag of Words

The easiest and the most common way to create a numerical representation of a text of arbitrary length is the *Bag of Words* abstraction. This method completely scrambles word order in texts and only cares about words[27] used in the text and their intensity, representing it in a dictionary data structure manner, with keys being words and values being numbers they are used within the text (2.11). Texts are usually stemmed and de-capitalized before applying Bag of Words, to make different forms of a word[28] stack together in one dictionary entry.

(2.11)
The quick brown fox jumps over the lazy dog.
{the:2; quick:1; brown:1; fox:1; jumps:1; over:1; lazy:1; dog:1}

To move from dictionary to vectorized representation for the Bag of Words, the final scope of the analysis is decided, all dictionaries for all texts within the scope are added together to form the ordered corpus dictionary, and every entry word within the corpus dictionary gets a ***one-hot encoding*** which represents it.

[27] The method can actually be used for any representation layer of the text: it is possible to create a Bag of Characters of a Bag of POS-tags
[28] Without it *'Word'*, *'word',* and '*words'* would be put in three different dictionary entries and the model would treat them as completely different words

One-hot encoding for the entry **k** in a dictionary of **N** words means, that the word **k** is represented with a vector of length **N** with **1** in k-th position and **0** in all other positions. To form vector of a text within the corpus following the one-hot encoding procedure all one-hot vectors for all words within that text are simply summed together, creating a vector of the same length with word intensities for this particular text in their respective one-hot positions of the encoding.

One of the most limiting factors of this approach is the dependence of the model dimensionality on the size of the dictionary. It creates problems if new texts with unknown words are added within the scope of the analysis, as each new word added to the dictionary changes the dimensionality of the one-hot representation, which means all vectors for all documents in the corpus have to be updated.

If there is a possibility that new texts will be analyzed with the representation and there is confidence that corpus dictionary is rich and new words occurrence would be quite rare even in unknown texts, it is possible to fix the dimensionality of the model with an **'_unknown_'** pseudonym for rare words. For that all frequencies of all words in the corpus are analyzed and all rare words with frequencies below some manually chosen threshold are removed and replaced with **'_unknown_'** pseudonym. Consequently, all texts are re-processed in a way, that every word not in the dictionary is considered **'_unknown_'** and adds to the **'_unknown_'** frequency instead of having its own. This method allows to process completely new texts with the same fixed dictionary, as long as there is confidence that unknown words would not be too common.

Another problem with the Bag of Words text representation is that it treats all words equally, which makes most frequent words in a text its most prominent features, even though many of the most frequent words in all texts are Function Words such as "the", "is", "an", "that" etc. for English, which have very little lexical meaning and do not represent any text very well[29].

This creates most problems in tasks which involve comparison of different texts to each other through their vectorized representations, as the presence of the same frequent stop words in both compared texts can make their vectors look more similar than these texts actually are. There are three common ways to offset this problem:

1. Create a list of stop words for the language, remove them from the dictionary and completely ignore them in processed texts.
2. Set a fairly low frequency limit for a word in a text, so any word could not have frequency higher than the maximum value.
3. Use modified tf-idf values in vectors instead of real word frequencies which are calculated through the algorithm which automatically downscales the importance of a word in a text based on how common this word is in the dictionary.

[29] These words are often called "stop words" in text processing

Approach 1 is effective but restrictive in a sense that it completely removes some words from the model even though they are represented in the text, and is usable only for semantic analysis on a word level (as there are no conventional 'stop letters' or 'stop parts of speech').

Approach 2 is less restrictive, as it only removes word entries that above the threshold in frequency, but requires additional analysis of each particular model because the threshold parameter is really important here and can severely damage representativeness of a model if set too low or have no effect if set too high.

Approach 3 is the most complex one and requires additional computations, but provides the best results for tasks which involve comparison of document vectors such as document retrieval, document classification , and document clustering. In this project the tf-idf method is used on a sentence level in the sentence alignment task which is used to detect and extract edited sentences from a pair of consecutive Wikipedia revisions (more in 3.4.2).

2.2.3.2 Tf-idf

Tf-idf technique stands for "term frequency - inverse document frequency" and involves scaling term frequency values which are used in the Bag of Words representation with special "inverse document frequency" multiplier which is calculated for each word in the corpus and stays the same for each document. The idea behind the inverse document frequency value is to represent the commonality of the given term within the given corpus of documents: it has to be low for terms which are present in many documents and high for those that are present only in some small number of documents. It was introduced in [47] as "term specificity" and had a lot of proposed different ways of computing it since then. In this project the built-in tf-idf method for the GraphLab Create[30] is used, which computes inverse document frequency as $ln\left(\frac{N}{f(w)}\right)$ with *N* being the total number of documents in the corpus and *f(w)* - the number of documents the term *w* appears in. This method of idf calculations completely nullifies frequencies of terms which appear in every document[31], making it more generalized version of the **Approach 1** described earlier.

All methods of textual data representation described so far suffer from the problem of dimensionality: vector representations produced in them are extremely high dimensional in many cases, especially with word data, and are unstable in the number of dimensions, which is dependent on the size of the dictionary. Introduction of rare words pseudonym partially solves the latter aspect of the problem, but also limits the model with masking of known rare words, which might be really important for the task (and makes tf-idf really hard to use), and does not address the problem of high dimensions at all.

Another aspect of one-hot encoding which naturally feels 'wrong' is the sheer sparsity of resulting vectors: for a dictionary of only ten thousand words a vector for each word consists of nine thousand nine hundred and ninety nine zeros a single one. Naturally, these methods of textual data representation are called "Sparse Encoding" and are virtually unusable in

[30] https://turi.com/products/create/docs/generated/graphlab.text_analytics.tf_idf.html
[31] ln(N/N) = ln(1) = 0

many Machine Learning algorithms, especially Neural Network based, due to unachievable computational requirements. For example, if the Network takes a word-level trigram as its input, and the dictionary of the processed dataset consists of fifty thousand words - the input vector for the network will consist of one hundred and fifty thousand nodes, which is basically unprocessable for a big dataset, despite the fact that most values within these nodes are zeros, even if the first hidden layer is only 10 nodes wide.

2.2.3.3 Dense feature representation and word embeddings

Alternate solution which is called “Dense Encoding” was derived from the way neural Networks process provided sparse-encoded data. On the Figure 2.4 it can be seen, that in terms of interactions between layers of the network, each transition from one layer with **N** nodes to the next layer with **M** nodes represents functional projection of a vector within some N-dimensional space onto another vector within another M-dimensional space. This projection happens while trying to solve some predefined task, so by definition it has to preserve all information relevant to this task. In other words - thro layer transformation networks creates an **abstraction** which represents input data in transformed manner in which features relevant to the task are more distinct. From the input layer to the output layer network moves through consecutive abstractions of input data, gradually making it clearer for the task.

Thus, if the task for the data is defined as **predicting itself** - no information relevant to the data representation would be lost and each layer of resulting network would hold a different compressed dense representation of the input data. This principle is commonly used in autoencoders for any types of data, however for textual data it is currently common to take advantage of the theoretically motivated knowledge about the internal structure of texts in natural languages and set the task differently.

Most methods for deriving dense vector representations of word-level textual information rely on the assumption that, following Markov Assumption, words in natural texts can be reliably predicted by words in their immediate context, discarding all parts of the chain connection to the rest of the text [48]. Meaning that words which often appear in similar contexts share same semantic connections with other words in the language and ergo are semantically related and should be close to each other in the semantic space.

This leads to the reformulation of the task which motivates dense encoding: instead of predicting itself, words should predict other words from their immediate context, as the learning optimisation over this task should lead to good dense representations of vectors with vectors for semantically similar words being close to each other in the embedding space using mathematical definition of distance (normalized euclidean or cosine distance). There are two common ways to approach this task: to predict a word by its aggregated context [49], and to predict random words in the context using the target word as an input (Skip-gram approach) [49], [50].

The latter approach is used in the **word2vec** models of word embeddings popularized by Google researchers [49], [51]. To avoid the multi-class classification problem of prediction of

a particular word in the immediate context against all other words in the dictionary, the goal in the word2vec approach vas formulated as a binary classification problem: the model should maximize the probability, that the given pair word+context came from the training data (and not just a random combination). However, extracting only words and contexts from the data generates a training set which has only positive examples, so the model have no evidence that there are words which are unlikely to be in the same context together and creates a trivial solution where the model maximizes the probability by making all words very close to each other. In word2vec approach negative sampling from random distribution was added to offset this problem and artificially create word+context pairs which **did not** come from the training data [50].

Thus, for each word in the subsample of the data (a batch) the word2vec model operates as follows:

1. extracts positive examples of this word and a context word pairs, randomly sampled without repetition from the predefined context window;
2. generates negative samples by pairing this word with words randomly sampled from the whole dictionary;
3. randomly initializes vector representations of predefined size for new words in the batch;
4. trains the binary classification model over the batch of positive and negative examples, optimizing word vectors in the process;
5. saves the current state of word vectors (embeddings) and goes to the next batch, until it reaches the predefined number of steps. The model does not usually have an 'early stop' condition, as its optimisation task is auxiliary and does not matter.

If this process runs over sufficient amounts of data long enough, it generates word embeddings which exhibit desired attributional and relational similarities between vectors akin to those between words they represent. For example, top 5 nearest neighbours search in the pretrained word2vec model which is used in this project[32] for the word 'Oslo' gives Tromsø, Trondheim, Bergen, Stavanger, and Tønsberg (2.12), and the second nearest neighbour of the vector (*Oslo - Norway + Kazakhstan*) is 'Astana' which the capital of Kazakhstan (the nearest neighbor is 'Astan' which I assume is just a result of some cases of incorrect automatic stemming of the same word during the model training) (2.13).

(2.12)

```
for i in w2v_model.most_similar_cosmul(positive = [u'осло_S'], topn = 5):
   print ('%s %f' % (i[0], i[1]))
```

тромс_S 0.771052
тронхейм_S 0.760533
берген_S 0.751012
ставангер_S 0.745418
тенсберг_S 0.729850

[32] Model is trained and published by the RusVectōrēs project using Russian National Corpus and Russian Wikipedia together. The downdoad link can be found here: http://ling.go.mail.ru/dsm/en/about

(2.13)

```
for i in w2v_model.most_similar_cosmul(positive = [u'осло_S', u'казахстан_S'],
                                       negative = [u'норвегия_S'], topn = 5):
    print ('%s %f' % (i[0], i[1]))
```

астан_S 0.803319
астана_S 0.792972
узбекистан_S 0.777207
чимкент_S 0.767439
алма-ата_S 0.764438

These semantic relations exhibited by embedding models motivate their usefulness for word-level feature initialisation not just only for dimensionality reduction, but also as a knowledge transfer tool for narrow NLP problems with limited available datasets, which provides better initialisation, speeds up training and often improves resulting quality.

2.2.3.4 Continuous Bag of Words

The fact that simple element-wise arithmetic operations are capable of representing relational similarities as vectors within the same space also gives more justification for the simple method of handling textual inputs of arbitrary length with dense representations which is called Continuous Bag of Words (CBOW). CBOW is an extension of the simple Bag of Words described earlier, but here, instead of scrambling the order and creating a dictionary of used words and their frequencies, all vectors for all words in the input are just pooled together through summation or averaging, creating a vector within the same embedding space for inputs of any length.

Dense representations are used as the main type of features through this project. Externally trained word2vec model is used for word-level feature representation in classification experiments presented in Chapter 4. A model for creation of embeddings for characters commonly used in Russian texts is described in 3.4.1.4. These char-level representations are used as features in the final version of sentence boundary detector used in the extraction pipeline (3.4.1) and used as char-level features for representation of character level differences between two versions of same sentences (edit pairs) in classification experiments in Chapter 4.

2.3 Wikipedia Revision History as a source for mining naturally accuring text improvements: the study on the Wikipedia-based corpus extraction and manipulation

In this section I provide the theoretical motivation and the structure for the main goal of the study which is the development of a corpus of naturally-occurring stylistic edits in Russian. Relevant studies and some design decisions are also discussed here.

2.3.1 Previous research

The idea of mining Wikipedia for linguistic information is by no means novel or original, but the wast majority of works are oriented on the extraction of semantic relations through the internal structure of Wikipedia, and less concerned about actual texts. However, there are several papers which are extremely related to this study. Brief overviews of these are presented below.

Bronner and Monz use English Wikipedia revisions to extract a large set of user edits and later create a classifier which would distinguish between “factual changes” and “fluency edits, which improve the style or readability” [22].

They extract a data set of 923,820 user edits and create a manually annotated training subset of 2,008. 3 different ML classification algorithms are used on these data: Support Vector Machines, Random Forests, and Logistic Regression. All algorithms showed >85% accuracy on 10-fold cross-validation (87.14% best) and retained the accuracy of 85% while tested on a subset of unannotated manually data labeled by user comments [22].

The study by Bronner and Monz is the main inspiration behind this project. I will return to description of their design a lot in the following pages, especially in cases where the design of my study deter from thiers.

Max and Wisniewski used Wikipedia revisions to create a WiCoPaCo - Wikipedia Correction and Paraphrase Corpus for French. This work describes in great detail the process of edit extraction and cleaning, however they do not use any ML classification on the extracted data [31].

A unified 21-category taxonomy for features for extraction and classification of revision history edits is presented in Daxenberger and Gurevych. These features are based on differences between two versions of a document and include differences on metadata, text, language, and markup levels. The study includes a supervised machine learning experiment, which achieves an F1 score of 0.62 on a corpus of edits extracted from the English Wikipedia. The 2-category classification experiment was done in order to compare the performance to the classifier by Bronner and Monz and achieved 90% accuracy of a test set [32].

2.3.2 Argumentation for treating revision sequences of user edits as net improvements in texts quality

The idea that iterative rewritings of a text by random different people with different levels of topical knowledge, language proficiency, and different visions on what good writing is does seem far-fetched and controversial at the first glance. In fact, the eclectic style of longer articles formed as a result of many small-scoped edits is a one of the major arguments in the critique of Wikipedia[33].

[33] https://en.wikipedia.org/wiki/Criticism_of_Wikipedia#Quality_of_the_presentation

However, in general articles do seem to become better with time, which is why in most previous studies on Wikipedia revisions the question on whether user edits improve articles or not either wasn't addressed, or was treated as a theoretical assumption.

I believe, that it is actually possible to mathematically prove that revision sequences improve Wikipedia articles.

Let's consider an article **A**. How is it possible to know if **A** is an ideal article and does not require editing?
One way here is to:

1. Ask **every person** able to read the article if it is satisfactory.
2. If all answers are 'yes', then the article does not require editing and is the best it could be.
3. If some people answered 'no', the information on the possible improvements is collected from all of them, processed to form the best edit possible and it is applied.
4. Now to return to 1 and repeat until 2 is true.

This is not how Wikipedia operates. The algorithm in Wikipedia is as follows:

1. Ask **one random** person able to read the article if it is satisfactory.
2. If the answer is 'yes', then the article does not require editing now.
3. If the answer is 'no', the information on the possible improvements is collected and applied.
4. Now to return to 1 and repeat endlessly.

These two are textbook examples on the optimisation algorithms, the former being **the steepest descent algorithm** and the latter is the **stochastic descent algorithm with on-line learning.**

It is proven, that stochastic optimisation algorithms do converge to some delta area around the optimal solution given enough iterations [33], therefore it is plausible to state, that wikipedia articles improve with long enough series revisions. Also this proves, that **it is incorrect to treat each unique edit as an improvement**. This notion is very important to the application of a corpus of extracted edits.

2.3.3 Why we do not discriminate against robots

In the design of the edit extraction Max and Wisniewski in [31] write:

"We purposefully do not include revisions that were submitted by automatic bots, as we want to restrict the data to modifications that could be made by human contributors." [31, p. 2]

However, bots are designed by human contributors to cover the most obvious and common mistakes, and complete exclusion of these mistakes might be harmful to the resulting data.

Especially, if we consider, that the “mistakes of detecting mistakes” by these bots result in introducing unnatural mistakes to the text, **which are later corrected by human contributors and are collected into the data set as natural.**

Based on this, I think that automatic edits hold value and should be collected into the initial corpus. Through the manual frequency analysis of a random sample it is evident, that automatic corrections are easily identifiable by their unnatural frequencies and could be easily removed of normalized at any point of the corpus modification.

2.3.4 The -diff approach to edits’ extraction and why it is not used here

Another big difference between the design of this study and methods used in previous works ([22], [31], [32]) is renouncing the -diff command (which uses dynamic programming to search for the longest subsequence of elements that two analyzed lines have in common) as the main drive behind the methods for edits detection.

To detect edits Bronner and Monz use *-diff* the following way:

“Within our approach we distinguish between *edit segments*, which represent the comparison (diff) between two document revisions, and *user edits*, which are the input for classification. An edit segment is a contiguous sequence of deleted, inserted or equal words (...) A user edit is a minimal set of sentences overlapping with deleted or inserted segments.” [30, p. 357]

-diff approach was tested on the early stages of this study and has been deemed unfit for the role of the main tool in the edits detection task. It has some serious advantages:

- it allows to save computation on sentence splitting: sentence boundary detection is used only until the first detected boundary on the right and the first boundary on the left from the *edits segment*.
- It is quite easy to implement: there are *diff* commands in UNIX shells like **bash,** and easily obtainable libraries with *-diff* algorithm implementations are available for all popular programming languages (Text::Diff for Perl[34], difflib for Python[35] etc.).

However, the main problem with the *-diff* is that it is intended for comparison of homogenous non-segmented sequences of characters, and not for the processing of texts in natural languages. Sure, texts are also sequences of characters, by they become less and less homogenous as we increase the scope from a word to a sentence to a text, and there is no penalty for matching between different segments of the abstraction level in this family of algorithms.

One common way to mitigate the possible harm is to use *-diff* on a word level: creating an inner unstemmed dictionary for each pair of texts and treating each word form in the texts as

[34] http://search.cpan.org/~neilb/Text-Diff-1.44/lib/Text/Diff.pm
[35] https://docs.python.org/2/library/difflib.html

a unique atomic symbol for the purpose of applying the longest common subsequence search. This allows to avoid mismatching letters between words, but still leaves the problem of mismatching words between sentences.

(2.14)
A B C . D E F .
A B C . I D G H . J E F .
A B C . + I+ D+ + G+ + H+ + .+ J E F .

That is why *-diff* handles sentence insertion paired with existing sentence editing incredibly poorly. In (2.14) there is a simple example of how the *-diff* is able to mess up the edit detection in the case of the edit pattern quite common for the Wikipedia. Here the two-sentence text in the first line becomes the three-sentence text in the second line. **D** was changed into **J** in the second sentence of the first line, and another full sentence **"I D G H ."** was inserted between sentences one and two.

However, because the deleted from the sentence two symbol **D** happened to appear in the inserted sentence **"I D G H ."**, the *-diff* saves it, as it leads to the shortest edit distance and there is no penalty for matching between sentences, and instead of one sentence insertion and one sentence correction provides either two sentence corrections:

(2.15)
D -> I D G H
E F -> J E F

Or puts the second sentence of the result within the edit scope of the third, making the edit into the one big sentence correction:

(2.16)
D E F . -> I D G H . J E F .

Difference between (2.15) and (2.16) depends on the handling of the inserted stop symbols. In (2.15) inserted stop symbols are recognised at the step of the user edit scope detection, while in (2.16) they are not. Both approaches are unsatisfactory.

The former allows to isolate and ignore inserted sentences between two unchanged ones at the extraction step, but in cases like (2.14) provides (2.15): two absolutely misleading edits which not just add noise, but actively harm the corpus.

The latter approach is less destructive in handling the (2.14) misalignment problem, as it does not create subtle faulty edits, but just combines the sentence insertion and the rewriting into one, making it look like the original sentence was expanded and split into two. The problem here is that it handles all sentence insertions that way, even the most simple ones (2.17), (2.18).

(2.17)

A B C . D E F .
A B C . G I J . D E F .
A B C . + G+ + I+ + J+ + . D E F .

(2.18)
D E F . -> G I J . D E F .

This not only makes the post-processing for filtering out sentence insertions mandatory, but also makes it incredibly hard to implement. Misalignments like in (2.14) create fragmented edits (2.16), so it is not enough to just filter out user edits with edit segments consisting of complete sentences like in (2.18).

The only possible way to get rid of faulty misaligned edits is to filter out all extracted user edits with sentence boundaries within their edit segments, which is really restrictive and significantly limits the possible uses of the resulting corpus.

Another problem is that even if there is a method to remove all misaligned edits, the actual valuable edits within them (D E F . -> J E F . in (2.14)) are still unrecoverable, and as it was mentioned earlier, rewriting paragraphs while adding new sentences and editing existing ones simultaneously is one of the most popular practices in non-trivial Wikipedia editing, which is the most interesting.

Described above is one of the most obvious and easily recreatable problems from the *-diff* usage at the extraction step, and there are more obscure problems which arise in the real data. Let's consider a real Wikipedia edit as an example (translation here is irrelevant):

(2.19)
Город расположен на том же месте, где находился известный в древние времена город Амбракия. Также Арта известна своими фруктами, в частности, цитрусовыми.

(2.20)
Город расположен на том же месте, где находился известный в древние времена город Амбракия {основанной коринфянами в 640 г. д.н.э}. Амбракия была резиденцией базилевса Эпира Пирра, известного своим походом в Италию и своей "пирровой" победой над римлянами. В 189 г. д.н.э. городом овладевают римляне. С XII века город известен под своим нынешним именем. В 1203 г. захвачен норманнами. С падением Константинополя в в руки крестоносцев IV крестового похода, Арта становится центром одного из трех греческих государств - Эпирского деспотата. С XV в. в руках турков ,с кратковременным контролем венецианцев (1688 г.) и французов (1797 г.). Город учавствовал в всегреческом восстании 1821 г. и в последующих эпирских восстаниях ,но стал снова греческим только в 1881 г., по решению Берлинского конгресса. {Также} Арта известна своими фруктами, в частности, цитрусовыми.

In (2.20) a section of two sentences from a Wikipedia revision (2.19) was edited. The edit is complex and consists of:

- The (factual insertion in the first sentence (green)
- The stylistic deletion in the last sentence (red)
- Six sentences inserted between them (black)

(2.21) and (2.22) show how char-level and word-level *-diffs* see this edit respectively:

(2.21)

Г о р о д р а с п о л о ж е н н а т о м ж е м е с т е , г д е н а х о д и л с я и з в е с т н ы й в д р е в н и е в р е м е н а г о р о д А м б р а к и я - . -Т+ о+ с+ н+ о+ в а+ н+ н+ о+ й+ к- ж- е+ о+ р+ и+ н+ ф+ я+ н+ а+ м+ и+ + в+ + 6+ 4+ 0+ + г+ .+ + д+ .+ н+ .+ э+ . А+ м+ б р- т а- + к и+ я+ + б+ ы+ л+ а+ + р+ е з- в- е- с- т- н- а- - с- в- о и+ д+ е+ н+ ц+ и+ е+ й+ + б+ а+ з+ и+ л+ е+ в+ с+ а+ + Э+ п+ и+ р+ а+ + П+ и+ р+ р+ а+ ,+ + и+ з+ в+ е+ с+ т+ н+ о+ г+ о+ + с+ в+ о+ и м+ + п+ о+ х+ о+ д+ о+ м+ + в+ + И+ т+ а+ л+ и+ ю+ + и+ + с+ в+ о+ е+ й+ + "+ п+ и+ р+ р+ о+ в+ о+ й+ "+ + п+ о+ б+ е+ д+ о+ й+ + н+ а+ д+ + р+ и+ м+ л+ я+ н+ а+ м+ и+ .+ + В+ + 1+ 8+ 9+ + г+ .+ + д+ .+ н+ .+ э+ .+ + г+ о+ р+ о+ д+ о+ м+ + о+ в+ л+ а+ д+ е+ в+ а+ ю+ т+ + р+ и+ м+ л+ я+ н+ е+ .+ + С+ + Х+ I+ I+ + в+ е+ к+ а+ + г+ о+ р+ о+ д+ + и+ з+ в+ е+ с+ т+ е+ н+ + п+ о+ д+ + с+ в+ о+ и+ м+ + н+ ы+ н+ е+ ш+ н+ и+ м+ + и+ м+ е+ н+ е+ м+ .+ + В+ + 1+ 2+ 0+ 3+ + г+ .+ + з+ а+ х+ в+ а+ ч+ е+ н+ + н+ о+ р+ м+ а+ н+ н+ а+ м+ и+ .+ + С+ + п+ а+ д+ е+ н+ и+ е+ м+ + К+ о+ н+ с+ т+ а+ н+ т+ и+ н+ о+ п+ о+ л+ я+ + в+ + в+ + р+ у+ к+ и+ + к+ р+ е+ с+ т+ о+ н+ о+ с+ ц+ е+ в+ + I+ V+ + к+ р+ е+ с+ т+ о+ в+ о+ г+ о+ + п+ о+ х+ о+ д+ а+ ,+ + А+ р+ т+ а+ + с+ т+ а+ н+ о+ в+ и+ т+ с+ я+ + ц+ е+ н+ т+ р+ о+ м+ + о+ д+ н+ о+ г+ о+ + и+ з+ + т+ р+ е+ х+ + г+ р+ е+ ч+ е+ с+ к+ и+ х+ + г+ о+ с+ у+ д+ а+ р+ с+ т+ в+ + -+ + Э+ п+ и+ р+ с+ к+ о+ г+ о+ + д+ е+ с+ п+ о+ т+ а+ т+ а+ .+ + С+ + Х+ V+ + в+ .+ + в+ + р+ у+ к+ а+ х+ + т+ у+ р+ к+ о+ в+ + ,+ с+ + к+ р+ а+ т+ к+ о+ в+ р+ е+ м+ е+ н+ н+ ы+ м+ + к+ о+ н+ т+ р+ о+ л+ е+ м+ + в+ е+ н+ е+ ц+ и+ а+ н+ ц+ е+ в+ + (+ 1+ 6+ 8+ 8+ + г+ .+)+ и ф р+ а+ н+ ц у- к- т+ з+ о+ в+ + (+ 1+ 7+ 9+ 7+ + г+ .+)+ .+ + Г+ о+ р+ о+ д+ + у+ ч а+ в+ с+ т+ в+ о+ в+ а+ л+ + в+ + в+ с+ е+ г+ р+ е+ ч+ е+ с+ к+ о м+ + в+ о+ с+ с+ т+ а+ н и+ и+ + 1+ 8+ 2+ 1+ + г+ .+ + и+ + в+ + п+ о+ с+ л+ е+ д+ у+ ю+ щ+ и+ х+ + э+ п+ и+ р+ с+ к+ и+ х+ + в+ о+ с+ с+ т+ а+ н+ и+ я+ х+ ,+ н+ о - в+ с+ т+ а+ л + с+ н+ о+ в+ а+ + г+ р+ е ч- а+ е с+ к- т- н- о- с- т и+ м+ + т+ о+ л+ ь+ к+ о+ + в+ + 1+ 8+ 8+ 1+ + г+ . , + п+ о+ + р+ е+ ш+ е+ н+ и+ ю+ + Б+ е+ р+ л+ и+ н+ с+ к+ о+ г+ о+ + к+ о+ н+ г+ р+ е+ с+ с+ а+ .+ + А+ р+ т+ а+ + и+ з+ в+ е+ с+ т+ н+ а+ + с+ в+ о+ и+ м+ и+ + ф+ р+ у+ к+ т+ а+ м+ и+ ,+ + в+ + ч+ а+ с+ т+ н+ о+ с+ т+ и+ ,+ ц и т р у с о в ы м и .

(2.22)

Город расположен на том же месте , где находился известный в древние времена город Амбракия - .+ основанной - Также+ коринфянами+ + в+ + 640+ + г+ + .+ + д+ + .+ + н+ + .+ + э+ + .+ + Амбракия+ + была+ + резиденцией+ + базилевса+ + Эпира+ + Пирра+ + ,+ + известного+ + своим+ + походом+ + в+ + Италию+ + и+ + своей+ + "пирровой"+ +победой+ + над+ + римлянами+ + .+ + В+ + 189+ + г+ + .+ + д+ + .+ + н+ + .+ + э+ + .+ + городом+ + овладевают+ + римляне+ + .+ + С+ + XII+ + века+ + город+ +

известен+ + под+ + своим+ + нынешним+ + именем+ + .+ + В+ + 1203+ + г+ + .+ + захвачен+ + норманнами+ + .+ + С+ + падением+ + Константинополя+ + в+ + в+ + руки+ + крестоносцев+ + IV+ + крестового+ + похода+ + , Арта + становится+ + центром+ + одного+ + из+ + трех+ + греческих+ + государств+ + -+ + Эпирского+ + деспотата+ + .+ + С+ + XV+ + в+ + .+ + в+ + руках+ + турков+ + ,+ + с+ + кратковременным+ + контролем+ + венецианцев+ + (+ + 1688+ + г+ + .+ +)+ + и+ + французов+ + (+ + 1797+ + г+ + .+ +)+ + .+ + Город+ + учавствовал+ + в+ + всегреческом+ + восстании+ + 1821+ + г+ + .+ + и+ + в+ + последующих+ + эпирских+ + восстаниях+ + ,+ + но+ + стал+ + снова+ + греческим+ + только+ + в+ + 1881+ + г+ + .+ + ,+ + по+ + решению+ + Берлинского+ + конгресса+ + .+ + Арта+ известна своими фруктами , в частности , цитрусовыми .

Both char-level and word-level approaches presented in (2.21) and (2.22) fail to recognise the quite straightforward structure of the editing and return the unrecoverable mess instead. In both these examples *-diff* completely fails to separate inserted sentences from the two perfectly suitable sentence-scoped edits and returns a single user edit with the scope covering the whole eight-sentenced segment instead.

It is important to point out, that this example does not fall down under the structure described in (2.14) and, actually, is much more concerning. For example, the word “Арта” in the second sentence of (2.19) actually does not changes through the editing process and stays exactly at the same position in (2.20), yet in both (2.21) and (2.22) it is not recognized as a part of the common subsequence between edits and misaligned.

Also, the dot at the end of the first sentence of (2.19) is not preserved but deleted and inserted later in both (2.21) and (2.22) for reasons I can not explain, which further reinforces the suspicion to this approach.

With all these problems the *-diff* has with the sentence boundaries it seems like the most restrictive postprocessing is the only way to protect the corpus from misaligned edits created by the *-diff* itself, but by applying restrictive postprocessing we effectively throw away most of the work and computation we invested in the edit extraction, which makes the whole design very questionable.

In [22] Bronner and Monz implemented the (2.16) approach and applied restrictive postprocessing similar to the one described here. They removed all complete sentence insertions and all fragmented user edits (with two or more edit segments in the same user edit), which resulted in filtering out of 74% of all extracted user edits [30 p. 5].

All these problems and concerns create a strong case for doubting the -diff approach to edits extraction and support the incentive to try something different.

2.3.5 The alternative method of extraction

In 2.2.4 I stated that most problems with using the text-wise Longest Common Subsequence approach with *-diff* algorithms rise from inability to punish the intersentential subsequence alignment, so the algorithm would use it only if all other alignment possibilities were substantially worse.

Even if it is possible to rewrite the LCS algorithm from scratch with addition of weights based on the number of potential sentence boundaries between aligned elements, but this approach seems unnecessary demanding and complex, as it would require substantial amounts of development and testing time, including the construction of specialized testing datasets to fine-tune (or train) the exact weight numbers, so other alternatives were explored.

Given that the restrictive post-processing seems to be the only way to reliably avoid potential faulty edits generated by the intersentential misalignments, and the post-processing described in 2.2.4 and used by Bronner and Monz discards all extracted edits which span over more than one sentence, it is reasonable to explore the approach that would make both compared consecutive revisions sentence-separated **before** the actual detection of differences. This would make possible to proceed with the extraction of user edits directly on the sentence level, using a similarity metric which would differentiate between identical (unchanged) sentences, similar (edited) sentences, and different (unrelated) sentences.

With this approach the extraction algorithm would have to perform following steps:

1. Detect all true sentence boundaries within both revisions and split both texts into sentence-separated versions.
2. Compare all sentences from the first revision to all sentences of the second revision using the predefined similarity metric and detect sentence pairs which are not exactly the same but very similar in meaning and relative position in the text and very likely belong to the same user edit.
3. Extract these sentence pairs as user edits and save them into the corpus.

This approach has the obvious potential performance chokepoint in the step 2: comparison of all sentences of revision 1 with all sentences of revision 2 scales exponentially with the length of the revisions, but as this operation can be represented as a matrix multiplication, it has very high parallelisation potential and should not be a problem for the modern computers if the chosen sentence similarity function is fast.

2.3.6 Language independency and modularity

Maximum language independency is the crucial point of the design philosophy for this project. As it was discussed earlier, Wikipedia maintains its structure between versions in different languages, therefore it is practical to develop processing tools which could be easily adjusted between languages.

To be language-independent, the processing pipeline should use minimal amounts of rule-based language-specific modules, and if these modules are used, they should be handled within the code with extreme caution, so they could be easily replaced by an alternative for the different language in the transition.

Therefore, as many modules as possible should be data-driven and be able to extract required linguistic knowledge from raw or weakly processed corpora of the target language texts. Modules which require prior training on additional language information should be uploaded into the pipeline through clear entry points and not trained within.

Despite being outside the extraction pipeline, the classification algorithm for stylistic and factual edits also should not rely on features engineered with the model-driven approach specific to the language, so it could theoretically be attached to the pipeline if needed without compromising its design.

Chapter 3

Data extraction

3.1 Final structure of the processing pipeline

Following the design decisions decisions described in 2.3, the final architecture of the processing pipeline is:

1. **Extraction and dewikification module:** directly works with the Wikipedia markup structure, extracts main texts of revisions, clears it from the Wikipedia internal markers (semantic tags, internal links, embedded images etc.). **Takes Wiki articles, returns raw rexts and additional meta information if needed.**
2. **Sentence splitter:** processes the extracted text, detects sentence boundaries within it, and adds the newline symbol after. **Takes raw texts, returns sentence-separated texts.**
3. **Sentence aligner and edits extractor:** takes two texts and tries to align them sentence by sentence. Based on the similarity metric between sentences and the distance between counting numbers of considered sentences in their texts. Sentences are deemed similar, completely different, or edited versions. Pairs of edited versions are extracted. **Takes pairs of sentence-separated texts, returns set of pairs of edited sentences (edits).**

Classification module is detached from the pipeline for testing and tuning purposes. It **takes a corpus of extracted edits and returns a corpus of stylistic edits**.

Because of the gigantic size of Russian Wikipedia, initial design of the data extraction module assumed a web mining approach. Article revisions was extracted directly from the Web and saved on disk in organised batches. This approach was implemented, but later dropped due to its unsatisfactory speed, reliance on the Internet connection and mysterious Web-related bugs.

3.2 Local processing: the problem of dump sizes

Local processing was chosen as an alternative to the web mining, so the problem of dump size of Russian Wikipedia with full revision history had to be addressed. The dump of January 11 2016 was used in this study. It consists of 4 files, which take 17 GB of disk space achieved in .7z format and about 2 TB unpacked. Files contain the full revision history of Russian Wikipedia in the .xml format.

Due to impossibility of processing of unpacked files on a home computer, I decided to implement a streaming approach: 7z is called from the data preprocessing program and

pipes unpacked data to it. The following bash script is used, as it is necessary to suppress information messages from the output stream:

```
#!/bin/bash

7z e -so -y ~/Documents/wikidamps/ruwiki-20160111-pages-meta-history2.xml.7z 2> /dev/null
```

3.3 Using the DKPro JWPL for the preprocessing, dewikification and pipeline control management

Java programming language with the DKPro JWPL (DKPro Java Wikipedia Library) was used in the processing of the .xml data stream [34].

Overall design is done in a streaming fashion:

- xml data from the archive is piped to Java as a stream with 7z;
- xml is parsed with XMLStreamReader as a stream and each revision text with accompanying information is extracted as it is encountered in the XML;
- the extracted revision text is cleaned from the XML markup with JWPL;
- revisions of the same page that differ after cleaning are collected as pairs in memory to form a batch;
- as soon as the batch is complete, it is piped further to the Python sentence splitting and aligning script that finds and saves sentence pairs - user edits;
- when Python script finishes its work, control flow is switched back to Java, where edits batch is attached to the main file of the corpus and then the stream processing continues to form the next batch.

The Java program is presented in the *WikiStreamer.java* file.

PageRevision class is used to keep extracted revision information: text itself, revision ID, page ID and editor's commentary. Page IDs are used to ensure that only revisions of the same page are collected as pairs.

ProgressCounter class controls the overall size of the processing session (in revisions), and the batch size parameter. Batch size controls how many pairs should be collected to a batch before the processing in Python starts. Batched design is important for the proper managing of system memory and for the minimisation of potential information losses due to the pipeline malfunction at the Python processing step.

Calling Python from Java is done via a bash script *py.sh,* which activates the *tensor*flow_env environment where all required python libraries are installed and starts the main python script *split_and_align.py* which handles remaining steps of the edits extraction.

```
#!/bin/bash

#source "/home/mithfin/anaconda2/envs/tensorflow_env/bin/activate" tensorflow_env
```

```
/home/mithfin/anaconda2/envs/tensorflow_env/bin/python -u
/home/mithfin/anaconda2/docs/Wikiproject/under_ver_control/split_and_align.py -
```

Java process then waits for the python script to finish with the current batch. During the processing the python script *split_and_align.py* extracts discovered edits from the current batch and saves it into the *batch.csv* buffer. After the python processing is done, the Java process takes initiative back and attaches the extracted edits to the main file of the corpus - *Mined_edits_comments.csv.* After that the processing iteration of the batch is complete, and the extraction of the next batch starts.

3.4 Main processing modules

Main edit processing was written in the Python programming language (Python 2.7) due to the relative simplicity of the language and the large variety of natural language processing (NLTK[36]) and machine learning (GraphLab Create[37], TensorFlow[38], Theano[39] etc.) tools and libraries available for Python, many of which are scripting frontends to efficient backend implementations of computationally expensive algorithms in compiling languages like C.

3.4.1 Sentence splitter

After the preprocessing and dewikification is done, two revisions are stored as two plain texts in Russian. The following step in the extraction process, following the design presented in 2.2.7, is to detect all sentence boundaries within both texts and split them into sentences, so each sentence in the text would start on a new line and finish with the newline character '\n'.

The sentence boundary detection (SBD) task is one of the easiest tasks in Natural Language Processing with a lot of successful solutions with accuracies up to 99% reported throughout past 20 years [35]. Given that, the initial idea was to use an external SBD program assuming that most languages with developed Wikipedias have external SBD programs created and available on the Web and therefore the usage of an external language-specific tool would not narrow the language-independent approach too much, especially at the initial stages of development.

3.4.1.1 Evaluation of the available splitter for Russian

For these reasons, the Lingua::Sentence[40] splitter was initially chosen as the SBD tool for the project. It is a rule-based splitter written in Perl and is based on the scripts used to create the EuroParl parallel corpus[41]. The splitter supports 18 languages including English and Russian in its current implementation. In its work it actively relies on the Nonbreaking Prefix dictionaries stored in separate files for different languages.

[36] http://www.nltk.org/
[37] https://turi.com/products/create/
[38] https://www.tensorflow.org/
[39] http://deeplearning.net/software/theano/
[40] http://search.cpan.org/~achimru/Lingua-Sentence-1.05/lib/Lingua/Sentence.pm
[41] http://www.statmt.org/europarl/

The script *sent-sep.pl* was written in Perl in order to process the small initial sample of Russian Wikipedia revision histories extracted from the Web and manually analyze the results. While the overall quality of the output looked good, a lot of cases involving quoted speech or unstandardised abbreviations were misclassified resulting in incorrect splits of the processed text into sentences.

Following this analysis, it was decided to test the perform a controlled experiment on the measurement of the accuracy of the Lingua::Sentence splitter for Russian. For this experiment the OpenCorpora Project's[42] corpus of Russian was stripped of the .xml markup and turned into a sentence-separated corpus of 92966 sentences.

Due to the fact that not all entries of the corpus were proper sentences with the sentence boundary symbol at the end, the corpus was filtered, leaving it with 80217 proper sentences. After, all newline symbols at the end of all sentences were removed and the plaintext version of the corpus (*opencorpora_plain.txt*) was saved separately.

The script *sent-sep.pl* was used on the *opencorpora_plain.txt* file to process it with the Lingua::Sentence splitter. Results were saved into the *opencorpora_plain_lingua.txt* file of 75936 detected sentences.
Comparison of the original set of sentences with the set generated by the Lingua::Sentence splitter in the *Sentence Boundary Detection.ipynb* IPython notebook estimated the accuracy of the Lingua::Sentence splitter at 85.7%.

3.4.1.2 Comparison of the Lingua::Sentence performance with the basic internally developed splitter

Following the surprisingly mediocre results of the Lingua::Sentence splitter on the OpenCorpora set it was decided to seek for alternatives. However, at that stage of development the search for another SBD program for Russian did not provide any results. Another possible lines of advancement was to try and create the alternative sentence splitter by myself. This would provide the full control over its internal structure and the code could be written with the idea of possible adaptation for another language in mind.

The Machine learning approach was chosen for the splitter over the rule-based one following the general tendency in the NLP field to move towards the data-driven methods as better performing ones described in the theoretical part of this paper, and direct reports showing the superior performance of learned SBD algorithms over rule-based and knowledge-based ones [35].
In order to provide the the maximal adaptability of the algorithm between different languages it was decided to avoid language-specific features or features extracted with the usage of external NLP programs such as Part-of-Speech taggers, Dependency taggers or Named Entity detectors.

[42] http://opencorpora.org/

Initial approach was to use only shallow features to establish the baseline performance estimation for the ML-based splitter and compare it to the Lingua::Sentence. 5 features chosen for the baseline SBD model:

- Capitalisation of the previous word (binary)
- Capitalisation of the following word (binary)
- Length of the previous word (integer)
- Length of the following word (integer)
- Number of words to the previous detected boundary (integer).

(3.1) *This is the first sentence. This is the second sentence.*

For example, the feature vector for the boundary between two sentences in (3.1) would be [0, 1, 8, 4, 5]. As for the class labels, the label '1' was assigned to non-boundaries and the label '0' was assigned to true sentence boundaries, which is somewhat counterintuitive, but puts the emphasis on the detection of dots (exclamation marks, question marks etc.) which do not mark a sentence boundary.

With only 5 features a sophisticated model design would be unlikely to provide a significant boost in performance, so only simple model designs were considered for this feature set.

A Logistic Regression model and a small Neural Network with one hidden layer (10 nodes) were trained using randomly selected 80% of the Opencorpora set, and tested on the remaining 20%. The Graphlab Create Python Machine Learning Framework[43] was used for the data manipulations and the modelling. Feature extraction, data preparation and models training and testing are presented in the *Sentence Boundary Detection.ipnb* IPython notebook.

#	Model	Accuracy	Precision	Recall
1	Basic Logistic	88.3%	**100%**	44.5%
2	Neural Network	**91.8%**	95.1%	63.0%
3	Logistic with balanced weights	83.3%	60.4%	60.5%
4	Neural Network with balanced weights	89.7%	78%	**68.8%**
5	Neural Network trained on the Yandex corpus	88.95%	83.57%	57.38%

Table 3.1: Results for Logistic Model and the Neural Network model (shallow features) for the Sentence Boundary Detection problem

[43] https://turi.com/products/create/

Rows 1 and 2 in the Table 3.1 present the results of both models trained on the unmodified data. Both models show high accuracy, surpassing the accuracy estimation for the Lingua::Sentence Splitter, and excellent precision, meaning that almost cases classified as non-boundaries were correct. However, the recall metric is very low for both models: 44.5% and 63%, which indicates that even for the better performing Neural model more than a third of non-boundaries were misclassified.

These results mean that both models are strongly biased towards the true boundary decision, and will likely produce a lot of fragmented sentences if used in the pipeline for the creation of sentence-separated revisions, and the accuracy gain is not that high as compared to the Lingua::Sentence.

One attempt was done in order to fix the low recall problem. The analysis of the dataset of true boundaries and non-boundaries extracted from the Opencorpora texts showed that it is heavily biased towards the true boundaries (20.74% to 79.26%). It is expected to have more end-symbols as true sentence boundaries in a natural text, but this also could have been the reason why the learned classifier was biased towards true boundaries, resulting in low recall values.

To evaluate the effect of the unbalance both models were re-trained with enabling the **class_weights='auto'** training parameter. This option automatically rescales the weights of classes during training based on their representation in the training set, thus rewarding the model more for guessing the underrepresented classes right. Results for these models with adjusted weights are presented in rows 3 and 4 of the Table 3.1.

As expected, the weight adjustment improved recall values for both models to 63.0% and 68.8% respectively, however this was achieved at the expense of significant drops in both accuracy and precision. F1 scores were calculated for both Neural models to evaluate if the precision-recall was beneficial. F1 score for the model without the class weights adjustment was evaluated at 0.76 and at 0.73 for the model with the weights adjustment. This indicates, that the weight adjustment is not worth it if we are to value both precision and recall equally.

Another assumption about the reasons behind the relatively poor model performance was the insufficient amounts of training data. To check if the increase in the size of the training set helps, the model was re-trained using the Russian half of the Yandex Parallel Corpus[44] (1 million sentences) and tested on the same text set as previou models.

The best results were achieved by the Neural model without weights balancing (Table 3.1 row 5) and were actually slightly worse than the results of the same model trained on the smaller set from the same corpus. This indicates the following:

[44] https://translate.yandex.ru/corpus

1. The design limitations of the model with shallow features do not allow it to benefit from larger amounts of training information.
2. Yandex dataset and Opencorpora dataset are likely to be domain-unbalanced relatively to each other, which could explain the slight performance drop of the 5th model.

These experiments provide enough evidence that with the described basic approach to the feature extraction it is possible to train a model which would perform at least as well as the Lingua::Sentence splitter for Russian. However, experiments revealed that the performance gain is not very high and the approach has some heavy limitations, so it is unlikely to improve the performance further without resorting to the use of features such as Part-of-Speech or Named-entity tags which would require implementation of language-specific modules.

3.4.1.3 Symmetric sliding window with knowledge transfer for sentence boundary detection

Gathered results indicated that it was unlikely to improve performance of the SBD model without incorporating features motivated by the internal structure of language on a better level that numerical and categorical information about a potential boundary surroundings. Here, instead of adding random arbitrary chosen features to the current model it was decided to fundamentally change the feature extraction process and homogenize the set of features.

A single sliding window approach was chosen for following experiments with SBD modeling. In this approach a 'window' of **2k+1** characters slides through the text, with (k+1)-th character being the **focus** and all other **2k** characters being potential features. If the focus character is detected as a potential sentence boundary indicator (dot, exclamation mark, question mark), then **2k** non-focus characters are used as features to decide if it is a sentence boundary or not.

This approach could be implemented with automatic sparse one-hot encodings of characters in the text, as the dictionary for characters is not that long for most languages including Russian, and encounters with out-of-dictionary characters are quite rare and could be easily covered by introduction of the **_unknown_** pseudonym for rare characters (see 2.2.3.1). However, in the case of Neural Network modelling this is basically equivalent to random dense initialisation of character vectors, and many publications about the initialization of Deep Neural Networks suggest that unsupervised pre-training always allows to achieve results which are at worse similar to results with random initialisation [52], [53]. Thus, it was decided to skip the sparse initialisation experiments and move straight to initialization of character features with pre-trained vectors.

3.4.1.4 Char-level distributed representation

Word2vec algorithm for creation of dense word embeddings [49] (see 2.2.3.3 for description) was adapted to create pre-trained dense representations of characters for the sliding window

SBD-model. Russian half of the Yandex Parallel Corpus was used for the unsupervised training. Python code for creation of embeddings is presented in IPython notebooks *char-level data extraction.ipynb* and *char2vec.ipynb*. The code uses the TensorFlow library by Google is based on the tutorial code examples for word2vec[45].

Here, in *char-level data extraction.ipynb* Yandex Corpus file **'corpus.en_ru.1m.ru'** is loaded in Python and initially transformed in a sequence of words, without separation of punctuation signs and any other usual preprocessing steps. Each space character is replaced with '_' symbol to free up the space character as a separation sign between data points. This preprocessing step is somewhat questionable, as it intentionally erases boundaries between space and underscore characters, but the overall number of underscores in the dataset is 5738 in 137087473 of characters, so the influence of this simplification should be negligible and it significantly simplifies the code.

After that data indices are split in 100 batches and processed further to create character-separated data representation. Because of this part the code should have been run twice with insignificant changes: to process the first and the second half of batches separately, as the processing of whole data did not fit into the memory. In two consecutive runs two data files were formed and saved: *charset_spaces0-67507539* and *charset_spaces67507539-137087473* . The example (3.2) shows the beginning of the first file.

(3.2) Т а к о е _ р а з в и т и е _ х а р а к т е р а _ Г а р р и _ м о ж е т _ р а з о ч а р о в а т ь _ ч и т а т е л е й , _

Consequently, both files are loaded in *char2vec.ipynb* notebook into **chars1** and **chars2** variables, which are concatenated into one **chars1** variable after. With that the dataset preparation processing is over - **chars1** mimics the structure of space-separated words collection which is used for positive sampling in the original word2vec algorithm (see 2.2.3.3). This allows to reuse the following code with minimal changes, mostly adjusting model parameters to account for the differences in word-level and character-level text structures.

The **build_dataset** function, which is defined and used at the following step, processes the **chars1** data using the predefined vocabulary size and creates four different data structures which describe data from **chars1** in more compact and processable way. The size 150 was chosen for the character dictionary in order to cover most of possible characters for Russian texts presented in the dataset but also leave some rarest characters uncovered in order to use them for training of 'unknown' pseudonym (**UNK**). **build_dataset** generates following data structures:

- **data** - a list of natural numbers in [0, 150] range, which represents **chars1** dataset mapped through the **dictionary.** Characters not from the dictionary are mapped to '0' which is the alias of '**UNK**' pseudonym.

[45] https://www.tensorflow.org/versions/r0.10/tutorials/index.html

- **count** - saves frequencies of dictionary characters in the dataset.
- **dictionary -** provides mapping of characters to numbers.
- **reverse_dictionary** - provides mapping of numbers to characters.

Generate_batch creates a training batch instance of positive examples sampled from the dataset according to the skip-gram model parameters (2.2.3.3).

- **Batch_size** controls the total number of positive examples used in one step of the training.
- **Num_skips** controls the number of positive examples randomly sampled from each window.
- **Skip_window** controls the span of context window from which positive examples are sampled.

The final model used in the project was created with **batch_size** of 128, **skip_window** of 2, and **num_skips** of 4. Batch size was chosen mostly arbitrarily, skip window was chosen much smaller than it is recommended for word embeddings, as characters are much less likely to exhibit long-term dependencies not covered by Markov Assumption[46]. Number of skips was chosen based on the window span. As the sampling process is performed without repetition, 4 samples cover the whole window, eliminating randomness in the positive sampling process and providing full contexts of small span for every focus character in the training data.

Models with different parameters, including wider windows and non-saturating sampling, were tested but didn't provide the quality of embeddings generated with parameters described here.

The final model parameter which controls the embedding training process is the length of embedding vectors. Here it was chosen to be 64 - this is much lower than usually used in training of word-level embeddings[47], but more that it is usually chosen for character representation. For example, Kim et al. used 15-dimensional embeddings in training of character-level language models [54]. However, in bigger character-based models like in [54] convolutional layers over small vectors of character embeddings are often used, which gather additional structural information over character sequences during the main training, after the generation of embeddings. As the plan was to not only use these embeddings for Sentence Boundary Detection, but also in classification later, it was decided to experiment with large character embeddings first, and decrease dimensionality later, if training over vectors of this size would be too slow or too memory-demanding, as bigger embedding size might be beneficial for speed and performance of models using them [41].

Following part of the code creates a word2vec model according to the description in 2.2.3.3 using Tensorflow functions and runs in over 1000001 step (the number was chosen to be large enough, but not too large, so the whole training would run only for couple of hours). At each step a batch of positive examples is created from data using the **generate_batch**

[46] At least in prose
[47] Word level embeddings used in this project have dimensionality of 500

function described earlier and the current state of embeddings for this data are loaded from the model memory. Negative samples are generated automatically from the vocabulary during the model training with **tf.nn.sampled_softmax_loss()** Tensorflow function. 64 negative samples are used for every batch of 128 positive samples. It was decided to keep the size of negative samples small, as the vocabulary for characters is only 150 characters wide, as opposing to thousands of words usually present in word-level model vocabularies. This leads to increased chances of negative-sampling the same pair which is present as a positive example in the batch, which distorts the model.

After training character embeddings were saved into *char_embeddings_d150_tr1e6_w2_softmax_adagrad_spaces.csv* file. It was decided to not save a copy of the underscore embedding ('_') as the space embedding (' ') to avoid any possible problems with a csv cell of whitespace characters only. The 151'th entry for space embedding is added to the embedding dictionary as a copy of the underscore embedding each time after embeddings are loaded.

Several different visualisations for final embeddings were created after the training using t-distributed Stochastic Neighbor Embedding algorithm implemented in Scikit-learn[48]. This method creates a representation of a set of N-dimensional vectors in K < N dimensional space while trying to preserve as much of its internal structure as possible. One of 2-dimensional visualisations created this way is presented on the Figure 3.2

Figure 3.2 shows that the embeddings did indeed obtain internal structure which passes sanity check: Russian letters are separated from English letters, there is clear separation between uppercase and lowercase letters, numbers form an isolated cluster. For lowercase Russian letters, which are much more common in a Russian text than any other character, it is possible to see separation between vowels 'а', 'о', 'у', 'е', 'э', 'о', 'ю', 'я'[49] and consonants, and even within the lowercase consonants cluster there is a clear subcluster of sonorants 'р', 'л', 'м', 'н'.

As obtained character embeddings have distinct internal structure which agrees with theoretical knowledge, it was decided to stop further experimentation with embedding extraction and proceed with the Sentence Splitter construction.

[48] http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

[49]For some reasons 'и' is an exception and was projected way up in the plot, far away from the vowel cluster. What is interesting, English 'i' is also separated from the rest of English letters and also is projected at the top of the plot.

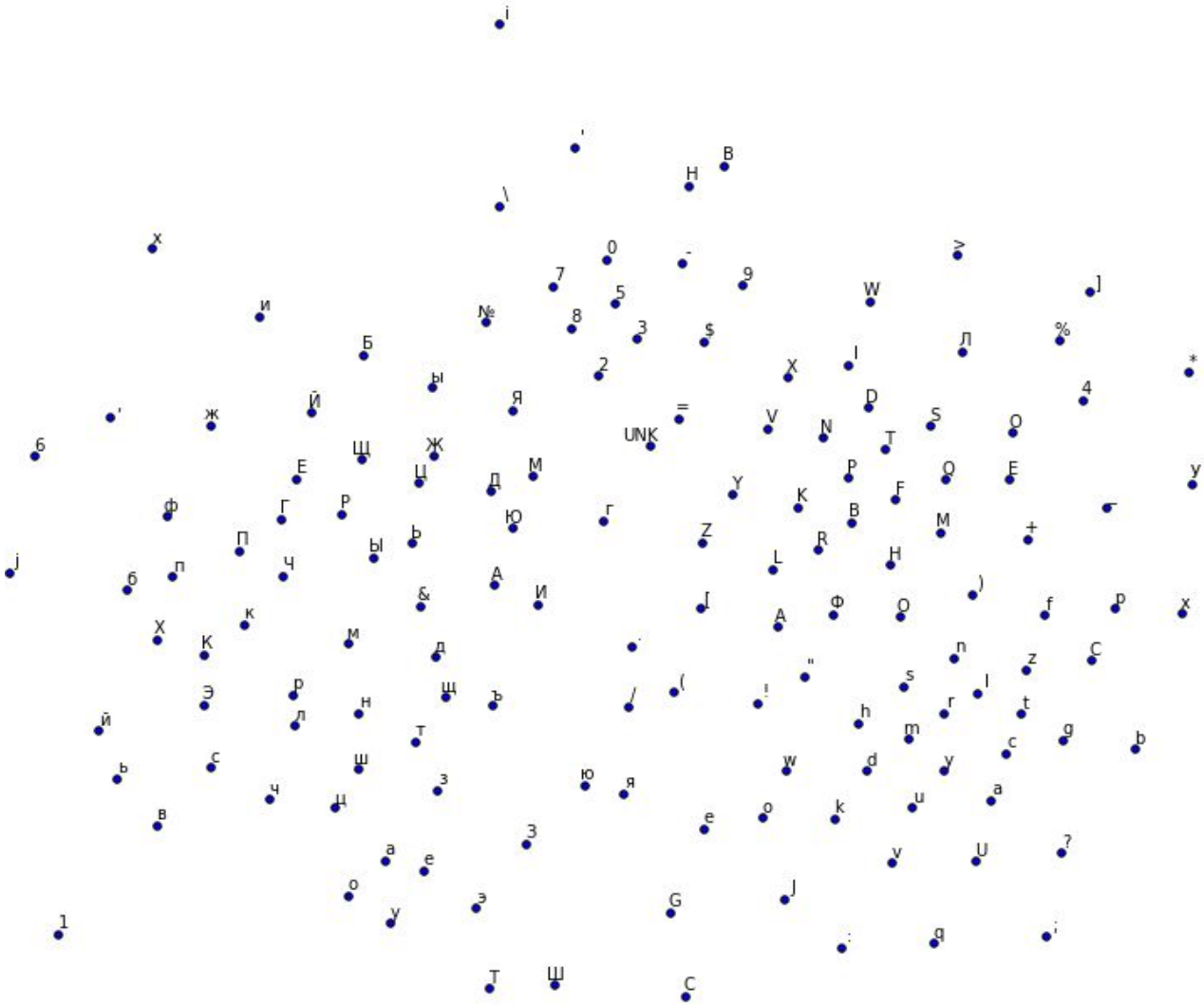

Figure 3.2: 2-dimensional representation of final 64-dimensional character embeddings.

3.4.1.5 Construction and evaluation of the symmetric sliding window splitter

The Python code for the construction and testing of the SBD model is provided in *Sentence Boundary Detection with char embeddings.ipnb* IPython notebook. Two different models were trained:

1. Smaller model with the sliding window of 15 characters (7 to the left and 7 to the right of the potential boundary) and the Feedforward Neural Network of two hidden layers (40 nodes and 10 nodes).
2. Bigger model with the sliding window of 17 characters and more complex Feedforward network (64 nodes, 32 nodes, 10 nodes). Dropout regularisation was added to the model to offset increased complexity.

Training was done on the 80% of the Yandex corpus (1010618 training examples). Validation testing for model tuning purposes was done using the remaining 20% of Yandex

corpus (228260 test examples). The final testing was performed on the whole Opencorpora corpus (98969 test examples). Result are presented in the Table 3.2.

#	Model	Validation accuracy (Yandex test set)	Validation recall (Yandex test set)	Test accuracy (Opencorpora test set)	Test recall (Opencorpora test set)
1	Window-15 model	97.95%	94.1%	**96.2%**	**91.3%**
2	Window-17 model	**98.04%**	**95.5%**	96.0%	91.1%

Table 3.2: Results for two symmetric sliding window character-based Feedforward Neural Network models for Sentence Boundary Detection trained with knowledge transfer.

Both models perform extraordinary well compared to previously tested models. While the bigger model performs slightly better on the validation set, it does worse on the test set, which might indicate that the initial performance gain was due to overfitting to the Yandex corpus, as both training and validation sets are created from it.

The increase in complexity and window span led to minimal gains on the validation set and to decreased performance on the test set, so it was decided to stop further exploration and proceed with the best performing model to the construction of the Splitter module. The model was saved as a GraphLab model under the name of *boundary_nn_model_r7_l40_l10_l2 .*

The final implementation of the Splitter module is written in Python and used as the Splitter class in the script *split_and_align.py* which is called from the *WikiStreamer.java* program (Chapter 3.3). Splitter loads the *boundary_nn_model_r7_l40_l10_l2* and moves the sliding window through the text, collecting features for all potential boundaries into the auxiliary dataset. After the data collection is done, the loaded model is used to classify all potential boundaries, and the end-of-line '**\n'** characters are inserted in the text according to the model decision.

To perform another evaluation of the splitter on the Wikipedia data and error analysis, a small subset of processed data was sampled and manually evaluated. Results of the evaluation are:

- Correct Sentences: 90.5%
- Data mistakes: 5.7%
- Algorithm mistakes: 3.8%

Evaluation showed that the percent of correctly detected boundaries was 90.5%, which is significantly lower than testing results. However, the error analysis showed, that significant part of incorrect boundary detections was due to mistakes in the data itself which lead to misclassifications. Error rate of the algorithms was evaluated at 3.8%, which gives about 96%[50] accuracy if data mistakes are excluded. This is consistent with test results and shows that the model generalizes well to Wikipedia data. Examples of data mistakes and algorithm mistakes are provided below:

(3.3) В переводе с тюркских языков означает "пять пальцев" **;.** В целом, блюдо представляет собой крошеное отварное мясо с лапшой…

(3.4) Хотя нужно отметить, что группа называлась "исторически" - King Diamon**d.3**атем был период, когда King был гитаристом в группе под названием "Brainstorm".

(3.5) В смысле классической логики логические связки могут быть определены через алгебру логики**.);** вообще не выполнит вызов подпрограммы some_condition если значение логической переменной action_required ложно, т.е. второй аргумент функции && вообще не будет опеределён.

(3.3), (3.4), and (3.5) showcase data mistakes. In (3.3) the random semicolon directly before the dot most likely mislead the classifier, as it is very unusual structure for the sentence boundary. In (3.4) the space between sentences was absent, which created a structure more common to nonbreaking dots. In (3.5) there possibly was a preprocessing dewikification mistake due to incorrect formatting of the revision, which removed a part of the sentence, leaving stray punctuation marks directly after the boundary, which lead to misclassification.

(3.6) С 1919 - член **ЦК РКП(б).** В 1918 - 1921 и 1922 - 1929 - председатель ВЦСПС.

(3.7) О популярности галушек говорит факт, что о них сложены песни, пословицы **и т.д.** В каждом крае рецепт отличался, но был более-менее общим.

(3.8) Ð, đ - буква латинского алфавита, сформированная добавлением поперечного штриха к вертикальной черте буквы D, **d.** Первоначально использовалась в средневековой латыни для обозначения сокращений, содержащих "д", например scđo для обозначения secundo.

(3.6), (3.7), and (3.8) showcase valid algorithm misclassifications which are consistent with the commonly acknowledged challenges of Sentence Boundary Detection - cases of structures which usually precede non-breaking characters (abbreviations, numbers, list indices etc.) being placed at the end of a sentence. In (3.6) it is the abbreviation “**ЦК РКП(б)**”, in (3.7) it is the very common abbreviation “**и т.д.**” (Russian version of “etc.”), and in (3.8) it is the English letter “**d**” at the end which is similar to non-breaking list index “**d.**”.

[50] 1 - 0.038 / (0.038 + 0.905) = 0.9597

3.4.1.6 Possible improvements

Overall, the performance of this module on properly formatted texts seems to be very good. However, Wikipedia Revision History seems to contain substantial amounts of data mistakes which decrease the efficiency of the Splitter and add noise to the extracted dataset. With that in mind, the most promising direction for the improvement of this module is to consider the shift from the consecutive classification of sentence boundaries to the classification of whole texts as *sequences of word sequences* divided by potential boundaries within one text-wide sequence-to-sequence model which evaluates all possible splits of the text into sentences and creates the best one. Or the preprocessing can be improved with identification of revisions corrupted by the dewikification process.

3.4.2 Sentence aligner

The Sentence aligner module is written in Python and used as the Aligner class in the script *split_and_align.py.* The aligner was programmed according to the model described in 2.3.5. Graphlab library is used for Machine Learning methods and algorithms. It gets who sentence-separated texts (one sentence per line) of consecutive revisions of some Wikipedia article. It then aligns sentences between these texts, distinguishing between exact match and fuzzy match. If the best alignment for a sentence is below the similarity threshold, the sentence is not aligned. All fuzzy matches below 100% similarity and above the threshold are considered to be user edits and extracted.

3.4.2.1 The NNS model for sentence alignment

Usually the sentence alignment task is performed for document in different languages and involve the usage of surface metrics such as number of words, length of words, relative position of the sentence etc. However, it this case two similar documents in the same language have to be aligned, which opens up the straightforward semantic approach to the task.

Following the idea of semantic alignment, the K-Nearest Neighbours Search (NNS) algorithm [55] which in NLP is usually used for document retrieval and recommendation was altered in order to create a model which would allow to align sentences between two revisions. The basic idea behind this approach is to treat the first revision as the set of queries and the second revision as the set of documents. In this setting, a metric which allows to calculate distance between a ‘query’ and a ‘document’ has to be introduced in order to perform the NNS search for each ‘query’ and find the Nearest Neighbour from the set of ‘documents’.

The most natural way here is to treat each sentence in the first revision as a ‘query’ and each sentence in the second as a ‘document’. This approach is implemented in this project,

as it was decided to stay within the scope of single sentence for the edit detection in order to minimise the chance of extracting faulty edits and maximize performance of the extraction pipeline. It is possible, however, to expand sets of 'queries' and 'documents' with sentence-level bigrams and even tri-grams in order to detect deep rewritings, if it is needed.

3.4.2.2 Levenshtein distance and performance concerns

As it was stated before, in this project the simplest approach to the creation of sets of 'queries' and 'documents' is implemented: each sentence in the earlier revision is a 'query' and all sentences of the following revision are 'documents'. This means that the distance metric has to measure the 'dissimilarity' between two sentences, with possibility to detect exact matches.

At the first glance, Levenshtein distance, which is the *"The smallest number of insertions, deletions, and substitutions required to change one string or tree into another."*[51] [56] seems to be the most natural way of measuring dissimilarity between two sentences in the case where it is expected to have a lot of exact matches. However, in this case it is imperative to find **the best possible** neighbour for each query, which is unusual for the NNS problem. This makes a lot of common algorithms used to speed up the search process unreliable, as they are often return the approximate best result, without any warranty that there are no better matches left in the data.

With that in mind, the brute force approach was implemented. It calculates pairwise Levenshtein distances between the query and every document and chooses the most similar one. The hope was that the generally small size of Wikipedia articles will allow this method to be fast enough to process Wikipedia Revision History in reasonable time. First experiments concluded that this is not the case: the algorithm ran for approximately 600 seconds on a single pair of moderately-sized revisions, which was far too slow.

3.4.2.3 Cosine distance and tf-idf weight adjustment

In order to speed up the alignment Levenshtein distance was discarded and cosine distance was used instead. All sentences from both revisions were added to one common corpus. For each sentence a Bag of Words representation was created and transformed using the Tf-idf adjustment over the conjoined corpus (in order to have the same idf coefficients for both revisions) (BoW and Tf-idf are described in 2.3.2). This approach to the sentence vector construction ensures that the same sentences in both revisions would have exactly the same representations, which results in cosine distance of zero.

It is important to point out that sentences were automatically stemmed before the BoW construction. This somewhat compromises the language-independent approach to the task, but stemming in one of the least complex linguistically motivated text transformations and

[51] Translation according to https://xlinux.nist.gov/dads//HTML/Levenshtein.html

currently stemmers are available for many different languages[52], so it was decided to proceed with the use of external stemmer for now. However, this is an obvious weak point in the integrity of the project and introduction of the aligning algorithm which would not require external programs should be among the top priorities for the future work on the extraction pipeline improvement. Currently the project uses **'mystem'** stemmer for Russian [57].

After the construction of Tf-idf representations for sentences in both revisions, brute force NNS algorithm with cosine distance is run to determine the nearest neighbour for each 'query' sentence from the first revision. This approach is much faster: the revision pair which took more than 10 minutes to process using Levenshtein distance was processed in less than a second.

3.4.2.4 Intratextual alignments and the differentiable decay function as a way to avoid them

Exploration of alignment results indicated that this approach has a serious flaw: it does not address positions of sentences in documents. Thus, if the same sentence was repeated twice in a revision one, and one of this repetitions was edited in the revision two, both copies of the sentence would align to the unedited sentence in the revision two, despite their relative positions in the text.

$$\frac{1}{1 + e^{|pos1 - pos2| - offset}} \quad (3.9)$$

To offset this problem a range decay coefficient was added to the model (3.9). It modifies the similarity between sentences based on the value of difference between positions of compared sentences in the text, and its form and span is defined by the *offset* parameter - which is the total difference between the number of sentences in revision one and revision two.

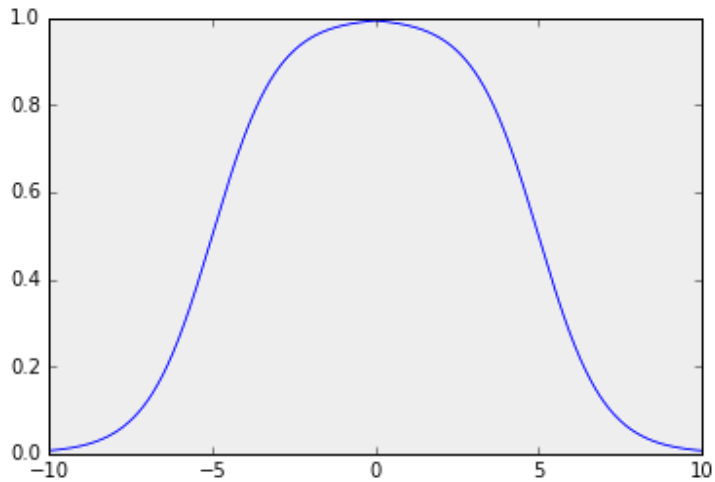

Figure 3.3 Shape of the range decay function

[52] http://www.nltk.org/api/nltk.stem.html, http://snowball.tartarus.org/

This range decay adjustments removes the possibility of accidental alignment of two sentences in completely different parts of the text, as it very unlikely that this alignment could have been a result of editing. In order to make this adjustments to not affect exact matches, instead of applying this function to the similarity, its inverted form was applied to the cosine distance metric, as the distance of zero between two exact matches is not affected that way.

Through empirical analysis of alignments the value of 0.8 was chosen as the heuristic upper boundary for the adjusted distance for two sentences to be considered edited versions of each other. Motivation and possible adjustment of this value is among the directions for the future work on the project.

3.4.2.5 Further performance optimisation

At this point the algorithm still was not fast enough, so additional optimisation was conducted. During the performance testing it was noticed that the algorithm calculates NNS and decay adjusting really fast, and spends the most time on the retrieval of edits from the final data structure. It was revealed, that SFrame data structure, which is used for data representation in Graphlab, does not handle multiple slicing really well and slows down the process significantly.

To fix this problem, it was decided to copy all relevant data from SFrame after the calculation of adjusted distances and extraction of aligned sentences, and continue processing using custom Python dictionary-based data structure. This improved the processing speed even further, improving from initial 600 seconds to 40 ms on the test revision pair.

3.4.2.6 Evaluation on the aligner performance on the set of manually processed edits

It was hard to reliably evaluate this alignment approach, as it requires very specific parallel data set of similar sentence-aligned Russian texts, and to my knowledge there it is impossible to find one in open access. Thus, the method was used without proper prior evaluation, but during the manual annotation of the the random sample of extracted edits additional marking category for **misalignment** was used in order to make an *a posteriori* estimate of aligner performance.

The manually processed set contains 3880 edit pairs, of which only 10 were marked as misalignment, which gives 0.25% error rate. This estimation does not evaluate the alignment algorithm fully, but indicates its very high precision for the edit extraction task.

Misalignments were analyzed in order to estimate possible weaknesses of the algorithm. Some of them are presented in Table 3.3. In the first example the Revision 1 of the document had English text, possibly copied from English Wikipedia to translate and mistakenly saved as a version of a Russian article. Text was translated and these sentences

were presumably aligned through the English letter ‘a’ which was used to represent a mathematical variable in Russian translation.

#	Revision 1	Revision 2
1	However, it should be noted that what is entailed is **a** solution to the general quintic.	Где **а** - комплексное число.
2	Отображает содержание **директории**.	Удаление **директории** вместе с файлами и другими **директориями**, которые она включает.
3	**С его точки зрения**, "говорящий субъект" в огромной степени предопределен самой структурой **языка**, от которой он отталкивается, и в зависимости от этой структуры качество "говорящего" и **содержание** "проговоренного" будут существенно меняться.	Структу́рная лингви́стика - языковедческая дисциплина, предметом которой является **язык**, изучаемый **с точки зрения** своего формального строения и организации **его** в целом, а также **с точки зрения** формального строения образующих **его** компонентов как в плане выражения, так и в плане **содержания**.

Table 3.3: Examples of misalignments in edit extraction.

Examples 2 and 3 follow the pattern: both are cases of significant rewriting of the article, where texts in Revision 1 and Revision 2 are different to the point that they can not be treated as different editions of the same text. Yet, both texts are describing the same thing and share the dictionary, so if by chance two sentences in approximately same positions in their texts share enough words from the common dictionary, they are calculated to be close enough and are aligned.

Currently the algorithm has no protection against cases like that, however, only 10 misalignments out of almost 4000 randomly sampled edit pairs show, that these situations are quite rare and should not affect the corpus significantly.

3.4.2.7 Possible improvements

Two main points of improvement for the Sentence Aligner module were already mentioned: it currently relies on external stemmer, which violates the language independence of the pipeline abstraction, and the number chosen to be the adjusted similarity boundary for extracted edits is weakly motivated and effects of this choice are mostly unexplored.

To address the first problem, some experiments on character-based CBOW representations of sentences for cosine similarity were tested and showed promising results. However, the problem of boundary between edits and unrelated sentences is even more glaring for that approach, and no sufficient solution was found in time.

One possible approach to solve both problems would be to use character-based CBOw with trained boundary value. The incredibly high precision of the current model, indicates, that the extracted set almost fully consists of real edits, so this set can be used to train the boundary value for the next iteration of the Sentence Aligner.

3.5 Pipeline performance

Final version of the pipeline described in this thesis functions autonomously from start (archived file of Wikipedia Revision History with files in Wikipedia markup format) to finish (file with extracted pairs of user edits). Pipeline processes one thousand of revision pairs in one to four minutes, depending on size of revision texts. In one month of continuous processing 7 millions of revision pairs (approximately 10% of the Russian Wikipedia Revision History) were processed, and about 2 millions of user edits were extracted.

Overall, performance of the Pipeline is acceptable to perform large scale research on the structure and properties of user edits in Wikipedia, and the size of extracted corpus is already sufficient for construction of edit-based models of text classification and text quality evaluation, or experiments on automated text editing.

The process of data extraction was halted to preserve the corpus in unchanged state for further classification experiments and free up processing power for classification modelling, as many of tested models are quite demanding on machine's operative memory.

The weakest point of the pipeline is definitely the alignment algorithm. Despite all the performance gains compared to the initial state, it still uses the brute force approach to the Nearest Neighbour Search algorithm, which handles big Wikipedia articles really poorly. Given that, exploration of precise alternatives for the brute force NNS or alternative sentence representations which would allow the use of LSH without misalignments should be the top priority for further development and improvement of the extraction pipeline.

Chapter 4

Edits' classification

4.1 Research motivation and important aspects of the experimental design

4.1.1 Is filtering needed?

As it was discussed earlier in 2.3.2, people do edits in Wikipedia to fix something they do not like, and thus make the text better[53]. We can make the assumption, that no matter the reason someone edits Wikipedia, they would not make the edit in **intentionally worse style.** New edits made by someone of course can be judged by others as being of worse style than the original, but this stands even for intentionally stylistic edits, as someone's understanding of good writing may be significantly different from others'.

Within this assumption it is reasonable to say, that then every Wikipedia edit can be considered as **non-worsening stylistic edit,** which means that stochastic reinforcement learning over all edits would still satisfy the goal of stylistic improvement, and consequently there is no need in further refining of the extracted set: it can be used for stylistic learning 'as is'.

This assumption is tempting, however even with it there is merit to filtering out edits which are less likely to hold stylistic relevance, as it condenses data in the corpus and makes training of complex models over it easier. Following this motivation the decision was made to not proceed directly to stylistic quality modelling over the extracted dataset and proceed with the structure outlined by Bronner and Monz [22] with exploring the possibilities of separating edits in the corpus into **factual** and **stylistic** classes.

4.1.2 Extracted dataset

2073249 user edits were extracted during the data extraction phase. 1148049 had a user comment attached, but only 77904 (~3.7%) could be reliably identified as style-related edits through these comments. This shows, that filtering approach using user edits shrinks data immensely, which is undesirable, as it is defeats one of the goals of the project - which is develop a method of extracting large quantities of style-related user edits.

It is likely possible to use these commented stylistic edits it training of a classification algorithm, but for that a reliable sampling of factual edits has to be constructed, which would

[53] There are vandalistic edits which do not follow this rule, but lately they are mostly detected by Wikipedia robots within minutes and reverted without trace in the revision history.

sample just factual edits without domain or type specification, which is, in a sense, a circular recruitment. Training of a model for classification between the set of commented edits and a random sample from the rest of the corpus performed terribly, so this approach was discarded.

Following the lack of success with user comments, I decided to proceed with the proposed in [22] approach of random extraction and manual classification of small subset of user edits in order to use it for data exploration as classification algorithm analysis. Dataset is randomly sampled over approximately 2 millions of extracted edits. The sample size was set at ten thousands and the initial plan was to annotate all of them, but due to time constraints the annotation was stopped at 3880 edits. This annotated subset was used in all classification experiments.

4.1.3 The choice of the baseline

A number of different methods of baseline calculation were explored.

- **Hard baseline**, defined by the majority class in the binary classification problem.
- **Soft baseline**, calculated with method taken from [22]: Accuracy of a single Binary Decision Tree trained with Levenshtein edit distance as its 1 feature.
- **Linear baseline** for non-recurrent models: results of Logistic Regression trained on the same feature set.

Hard baseline provides the estimation of the lower boundary for the classification accuracy, Soft baseline indicates the accuracy of the most simple algorithm for the task, and Linear baseline provides information on the linear separability of the data given the chosen set of features, which is useful for quality estimation of the chosen feature set.

4.1.4 Feature selection with distributed language representation

As opposed to [22], here, following the concept of language independency of the processing algorithm, which is needed for making the processing easily adaptable to Wikipedia data in another language, it was decided to avoid the use of linguistically motivated features in classification.

Thus, features used for classification in this module are all derived from raw language data through unsupervised learning. Algorithms use Character-level features which are presented by the same character embeddings used in the Sentence Splitter module and described in 3.4.1.4 and Word-level features which were not trained during this project but were downloaded as a pre-trained model from the open web storage of the RusVectōrēs project[54].

Word-level embeddings saved in this model were trained using data from Russian Wikipedia (the main reason this particular model was used) and Russian National Corpus. The model was trained using CBOW approach, which differs from the Skip-gram algorithm described in

[54] http://ling.go.mail.ru/dsm/en/about#publications

this project. In this method's auxiliary task a vector created from embeddings of words within the context window span using CBOW pooling (2.2.3.4) is used to predict the embedding of the focus word.

The model was trained with embedding vector length set to 500 and the context window span set to 2. Model stores embeddings for 604 043 different lemmas, infrequent words were replaced by **UNKNOWN_UNKN** pseudonym during training. Automatically derived Part of Speech tags were used in addition to lemmas for word identification is the model. The description of the model does not state the intended reason behind the addition of PoS tags to word identifiers, but the most logical explanation would be an attempt at homonym disambiguation during training. PoS tags were generated using **'mystem'** - the same program which were used in this project for stemming in the Sentence Aligner module (2.2.3.2). In order to be able to queue the model for embeddings **'mystem'** was also used in this module to stem words and derive their PoS tags. In cases where the program was unable to predict the PoS tag, the tag **_UNKN** was used**.** The same tag is used in the model for the same purposes.

Character-level and word-level embeddings are used as independent features or pooled using summation or averaging to create CBOW representations of various-length features such as 'words, inserted in the sentence during the editing'.

Extracted features include 4 basic, which are CBOW vectors created by the mean pooling operation from the vectorised representations of inserted words (**inserted_words_vec**), deleted words (**deleted_words_vec**), inserted characters (**inserted_chars_vec**) and deleted characters (**deleted_chars_vec**). These features are context-independent and are based only on the edit segment elements. CBOW vectors retain the dimensionality of embedding vectors: 64 for character level and 500 for word level.

chars_vec_diff - arithmetic difference between deleted and inserted characters, attempt to artificially focus on the change in the sentences rather than on the structures of insertions and deletions.

Additional tested features include various attempts to incorporate context in the model through CBOW representations:

sentence_projection, **aligned_sentence_projection** - trigram-based mean-pooling of all characters in original and edited sentences respectively;

char_sentence_diff - arithmetic difference between the two previous features

conc_sentence_projection, **conc_aligned_sentence_projection** - alternation of the previous trigram-based char feature pair, internal trigram pooling changed from max-pooling to the simple concatenation

conc_char_sentence_diff - arithmetic difference between the two previous features

wbw_sentence_projection, **wbw_aligned_sentence_projection** - word-based mean-pooling of all characters in original and edited sentences respectively;

w_char_sentence_diff - arithmetic difference between the two previous features

Word-level features used in this model require stemming and PoS-tagging by external programs, which violates the language-independent concept declared for the project. Usually, word2vec models does not incorporate PoS tags, so this part is easily fixable with training another word-level using Wikipedia data only and not using PoS tags. Stemming however is mandatory and probably unavoidable. One possible solution is to discard word-level entirely and use only character level features. In order to evaluate the impact of this decision char-only models were trained at every step of classification experiments and their performance was compared to models using both char and word levels of data representation.

4.2 Processing of extracted subset of user edits

4.2.1 Manual classification scheme

Following Bronner and Monz [22], in order to create a reliable training set for classification experiments the extracted subset was manually processed and user edits in it were classified. During the classification process it became obvious that the simple 'factual' / 'stylistic' classification scheme would not be enough, as in some cases it forces to make the decision about the binary class on the spot, even if it is not obvious. With the simple classification scheme if later it was decided to change the initial intuition and classify some type of edits as factual instead as stylistic - the whole set of edits would had to be reprocessed.Because of this motivation the following classification scheme was created and used for marking of extracted edits:

- **0 - Noise.** Edit, created by a shred of Wikipedia markup remained within one version of the sentence and edited out in another[55]. These edits are created by users using Wikipedia markup incorrectly, which disrupts JWPL regular-expression based preprocessing, which often removes only parts of the incorrect markup. It is not clear how these edits should be classified in the binary classification. For now they are filtered out and not used in the classification at all. In future, most likely they would have to be collected in a separate training set and a postprocessing detector will be trained to detect and remove them from the corpus.
- **1 - Factual edit**. Edits which are without a doubt factual. Correcting changes, deletions or insertions in dates, names, description of events etc.
- **2 - Stylistic edit.** Edits which are without a doubt stylistic. Changes in sentence structure of the choice of words which do not alter its meaning, but make it more fluent and easy to read.

[55] Оксид азота(II) NO – несолеобразующий оксид азота.",
"thumb|right|Оксид азота(II) Оксид азота(II) NO – несолеобразующий оксид азота.

- **3 - Orthographic edit**. Correction of simple mistakes in word spellings. It is not entirely clear, that these changes should be classified as stylistic. Yes, the presence of spelling mistakes in texts are without a doubt do not add ‘style’ to them, but formally a spelling change often alters stem of the word, making it somewhat factual, especially in the case of Named Entities. In this project Orthographic edits are considered stylistic and used in the classification, but this decision might not be final.
- **4 - Complex edit.** User edit which are both stylistic and factual within the scope of the sentence. Again, it is not entirely clear if the focus should be on the factual or of the stylistic aspect of these edits. If the goal is to preserve as much stylistic edits as possible, but remove obviously factual - they should be stylistic, but if the goal is to remove as many factual as possible, to create a corpus with mostly stylistic edits - these complex edits should be treated as factual. In this project they are used in classification experiments as factual edits.
- **5 - Vandalism.** Most vandalistic edits are detected and reverted automatically within minutes, either without adding them to revision history, or with a comment about vandalism prevention. However, sometimes vandalistic edits are stay long enough to be edited out by other people, and these series of edits are stored in the Revision History. Even if it is clear, that vandalistic edit themselves should be detected and removed from the corpus, as they disrupt the main assumption about edits made with the intention to improve articles, it is not clear what has to be done with edits which revert vandalism back to normal - as they have the aspect of stylistic editing in them. In this project Vandalistic edits are not used in classification experiments.
- **6 - Misalignments.** Mistakenly aligned sentences which are not directly related through sentence editing process. These edits are discussed in 3.4.2.6
- **x - Attention marker.** Some edits, while being potentially classifiable, highlight important problems within the corpus structure and processing pipeline. These edits are temporarily classified with the **‘x’** marker to be easily found late. After the resolution of the problem the sentence highlights it gets correct marker according to the rest of the scheme.

4.5.2 The helper program

In order to speed up manual classification a helper program was written in Java (Figure 4.1). The program operates from the command line: it prints both versions of a sentence from different revisions in two consecutive lines and prints out the color-coded word-level -diff result of comparison of these two sentences in the third line, distinguishing between deleted words, added words, deleted with replacement, and inserted with replacement.

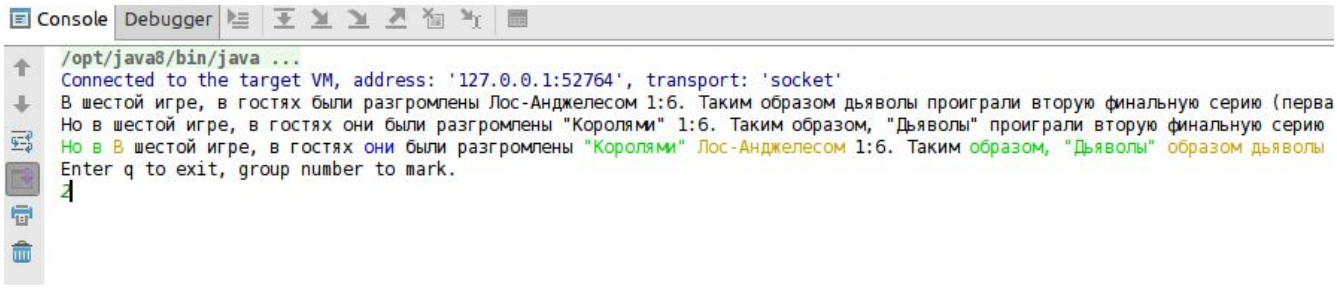

Figure 4.1: Screenshot of the *MarkingHelper.java* program in work

The correct marker based on the presented classification scheme is printed in the following line. The end-of-line symbol confirms the marker and adds it directly to the text file with user edits. After that the next pair of sentences is displayed.

This program significantly speed up the manual annotation process, and with minor additions could be used to involve additional annotators into the project with minimal requirements on their qualifications.

4.2.3 Overview of resulting structure

Table 4.1 shows the results of manual annotation process. Factual edits are the dominating majority with more than a half of all edits in the class. This result supports the intention of post-processing, if the final goal is to use the corpus for style-related research, as in the current state the sheer amount of factual edits could potentially disrupt any research on stylistic quality improvement using this corpus, even if they are stylistically neutral.

Stylistic, orthographic and complex edits are significantly represented, but pure stylistic edits amount to less that 20% of the subset, which is far less than expected. This, again, supports the need in further refinement of the corpus.

0 - noise	25 (0.7%)
1 - factual edit	1953 (50.3%)
2 - stylistic edit	743 (19.1%)
3 - orthographic edit	605 (15.6%)
4 - complex edit	336 (8.7%)
5 - vandalism	142 (3.7%)
6 - misalignment	10 (0.3%)
x - attention marker	66 (1.7%)

Table 4.1: Results of manual annotation

Noise and vandalistic edits seem to be rare. Even if they should be addressed at some point, these results show that they could wait.

For classification experiments it was decided to use Factual edits + Complex Edits as Factual and Stylistic Edits + Orthographic Edits as Stylistic. Thus, the resulting dataset is:

- **Dataset size -** 3637
- **Factual edits -** 2289 (62.94 %)
- **Stylistic edits -** 1348 (37.06 %)

4.3 Experiments with different classification models

4.3.1 The baseline model

All baseline models (4.1.3) were created and evaluated in *Edit Classification (baseline, linear, fully-connected).ipnb* IPython Notebook in the 'Hard Baseline', and 'Baseline Levenshtein distance classifier' sections.

Accuracy of the hard baseline is just a percentage of the biggest class presented in the data. Here is is the **Factual edits** class, thus the hard baseline for accuracy is **62.94%**

To evaluate the soft baseline Levenshtein distances for each user edit pair of sentences were calculated using *graphlab.distances.levenshtein* built in Graphlab method. As in [22], single Decision Tree (2.2.1.4) with one feature and depth one was used for modeling. Also a Logistic Regression (2.2.1.5) model was built for the same one feature in order to evaluate how the addition of distance weight affects the soft baseline model. Models were trained on 90% of the data and evaluated on 10% using random data split. Results are presented in the Table 4.2.

Classifier	Accuracy
Decision Tree	**79.2%**
Logistic Regression	76.5%

Table 4.2: Soft baseline accuracy estimation results

Soft baseline model scored incredibly high compared to the hard baseline, which is consistent with Bronner and Monz results for English - their baseline model performed at 76.34% accuracy. This result shows that stylistic edits are people are adding to the Wikipedia are usually shorter than factual, as results of a very simple model like this one, trained over small randomly sampled annotated subset could actually be used as an

heuristic in the processing pipeline, throwing out all edits of length higher than the model result, and this will significantly increase the share of stylistic edits in extracted data.

However, while simple and easily implemented, this method seems too restrictive, as there is an argument for those long stylistic edits to be the most interesting ones for research purposes, as they actually represent how people process and rewrite sentences to create better ones with the same meaning.

Results of Logistic Regressing over the same data supports this concern and indicates the presence of significant amount of incorrectly classified outliers in the data, situated far from the decision boundary in the opposite class' territory. This can be deducted from the almost 3% drop in accuracy: Decision Tree does not care 'how much' it is wrong in misclassified examples, while Logistic Regression is affected by distances to examples, and long incorrect distances are extremely punishing, which results in the decision boundary shift and accuracy drop caused by heavy outliers.

4.3.2 Random Forests

Relatively small dataset with lots and lots of features is ideal for the Random Forest classifier (2.1.2.4). It allows to use the main strength of the algorithm, which is the ability to extract the most from meaningful features and completely ignore the noise caused by ifeatures holding no information useful for the task, while avoiding its main weakness - extreme increase of training time with the size of training set.

Random forest classifier was trained and evaluated on the same 90%/10% data split as previous models. The model was trained in four versions. Four basic features (4.1.4) **inserted_words_vec**, **deleted_words_vec**, **inserted_chars_vec, deleted_chars_vec** (1128 unpacked features) and the same set of features + **levenshtein** feature for levenshtein distance (1129 unpacked features) for two-level representation. **inserted_chars_vec, deleted_chars_vec** (128 unpacked features) and these + **levenshtein** (129 unpacked features) for char-level . Results are presented in Table 4.3.

Feature Set	Accuracy
Char and word level CBOW edit features	84,9%
Char and word level CBOW edit features + levenshtein	**85.9%**
Char only CBOW edit features	**85.9%**
Char only CBOW edit features + levenshtein	**85.9%**

Table 4.3: Random Forests classification results

Other combinations of features described in 4.1.4 were tested, but none of them provided comparable results and they were not included in the final report.

This initial testing of the Random Forests classifier provided promising results, which are comparable to 87% achieved by Random Forests in Bronner and Monz. Especially noteworthy is the fact that due its algorithm this model is able to significantly benefit from the levenshtein distance feature without any additional normalisation procedures or other adjustments.

All models, however, significantly overfit training data with training accuracies exceeding 98% in both cases, which is a known problem for Decision Tree - based algorithms. This overfitting means that the resulting decision boundary is extremely jagged which may potentially result in sudden and unexpected performance drops and decision changes with slightest alterations in the representation of tested data. Therefore, the second model of this approach looks like a strong candidate for being the best performing one on the analyzed data, but full cross-validation is needed to confirm stability of its performance.

Models not using word-level representations performed exactly as the best model which uses them. This means that the word-level model is full of features insignificant for the task, which are almost never used in random decision trees construction due to its limited to 50 maximal depth. This is gives strong indications to the fact, that this particular word2vec model is not very helpful for the analyzed task. Is is unclear, however, is it due to this particular word2vec model being not very good, or does this mean that word-level features do not add additional information about this classification task to the information represented in character level ones.

4.3.3 Experiments with basic linear modeling

Basic Logistic Regression models were trained on all features to evaluate linear separability of their representation of analyzed data and compare with soft baseline results. Experiments were performed with the same train/test data split in the same *Edit Classification (baseline, linear, fully-connected).ipnb* IPython Notebook as all previously described models.

Extensive experimentation with different combinations of Char-level and Word-level features showed that again the best performing model is the one that avoids any attempts on CBOW context incorporation or direct vector manipulation and just uses 4 basic vectors of inserted and deleted characters and inserted and deleted words for the input. All inferior models was removed from this notebook in order to make it smaller and cleaner.

Feature Set	Accuracy
Char and word level CBOW edit features	75.4%
Char and word level CBOW edit features + levenshtein	**80.0%**

Char only CBOW edit features	76.3%
Char only CBOW edit features + levenshtein	79.1%

Table 4.4: Logistic Regression classification results

Linear models performed really bad, with the best model achieving accuracy of 75.4% using **inserted_words_vec**, **deleted_words_vec**, **inserted_chars_vec, deleted_chars_vec**, which is way below the soft baseline and even below the performance of single-feature linear levenshtein model. Addition of the levenshtein distance feature boosts Logistic Classifier performance to **80%,** which is slightly above the soft baseline.

Linear models were not expected to perform well, and were trained in order to evaluate the quality of different features extracted prior to modelling. Experiments showed that all invented methods of basic vector transformations aimed to emphasize the difference between feature vectors of two sentences and all attempts at context incorporation through CBOW resulted in much less linearly-separable features than the basic representation of edits through 1128-dimensional vector created through concatenation of 4 CBOW representations of inserted and deleted characters and words for two-level representation scheme and 128-dimensional vector created through concatenation of 2 CBOW representations of inserted and deleted characters only for one-level representation.

4.3.4 Feedforward Neural Networks

Experiments with Feedforward Neural networks was started in the same *Edit Classification (baseline, linear, fully-connected).ipnb* IPython Notebook as all models described before. Based on results from Random Forests and Logistic Regression it was decided to continue with only basic CBOW features, which continuously showed the best performance from now on, and focus more on testing of different Neural Network architectures.

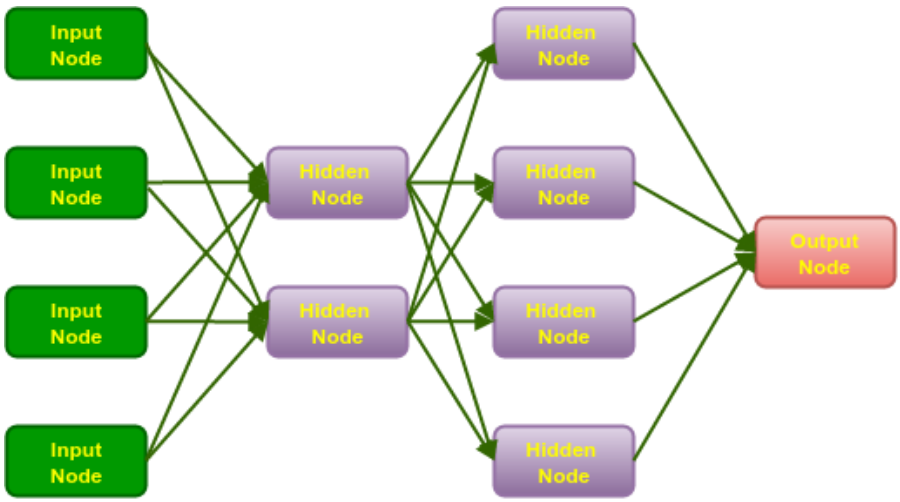

Figure 4.1: Narrow Filter example

Experiments showed that the best performing architectures have to involve narrow filters[56] (Figure 4.1) for Char + Word models and include dropout layers to prevent quick overfitting. For Graphlab-based modelling input data is also had to be rescaled with **_divideby_** and **_learning_rate_** parameters in order to push the model out of the local optimum of the majority class classification.

Output logs were analyzed and parameters for the number of iterations were chosen manually to promote the early stopping around the plateau of the optimal solution. Training and evaluation was performed using Graphlab library and the same train/test data split as before.

Feature Set	Accuracy
Char and word level CBOW edit features	85.7%
Char and word level CBOW edit features + levenshtein	81.1%
Char only CBOW edit features	**86.4%**
Char only CBOW edit features + levenshtein	83.1%

Table 4.5: Feedforward Neural Network classification results for GraphLab testing

As training trajectories for Neural Networks are much more unstable and tend to oscillate a lot, models were initially run for many iteration and logs were analyzed in order to indicate the performance plateau. Than models were retrained using early stopping at the plateau level. This method is useful to assess potential performance capacity of the given model, but does not indicate true performance, which has to be evaluated without explicit tuning for the data set. This evaluation will be presented later in the form of 10-fold cross validation with parameters unchanging between validation runs.

In this performance assessment the best validation accuracy values for Char+Word models seems to oscillate around 86.0% with the validation accuracy of 85.7% after the final 200th iteration.

Addition of the Levenshtein distance feature significantly **decreases** the performance of the model and regularizes it, preventing overfitting after any number of iterations. This means that some of informative dependencies learned by the Neural Network strictly contradict the bias of the levenshtein distance feature, thus creating the permanent training stalemate which results in worse accuracy but increased stability.

In this performance assessment experiment Char-only models did better than Char+Word models. The Char-only model's accuracy at the performance plateau oscillates around

[56] Narrow filter is a hidden layer of much lower number of nodes between two big hidden layers. It creates an 'information bottleneck' which forces the network to get rid of all all noise in the data and save only relevant information.This approach is widely used in image denoising: https://blog.keras.io/building-autoencoders-in-keras.html

86.5% with peaks higher than 87%. The model achieves accuracy of **86.5%** after the final 400th iteration.

Char-only model exhibits same behaviour as Char+Word model with addition of Levenshtein distance feature: both training and validation accuracy oscillate infinitely around same values close to 80%.

To test a theory about reasons behind word-level features hindering Machine Learning models another complex architecture was tested. This model was designed as two separate models, one for character level and one for word level, trained simultaneously on the same task, without exchanging any information up to the last layer (Figure 4.2). Both models have a single-node sigmoid layer at the end to fix the output in these nodes in [0, 1] interval. The Network makes its final decision based on these values through the softmax output layer for binary classification problem.

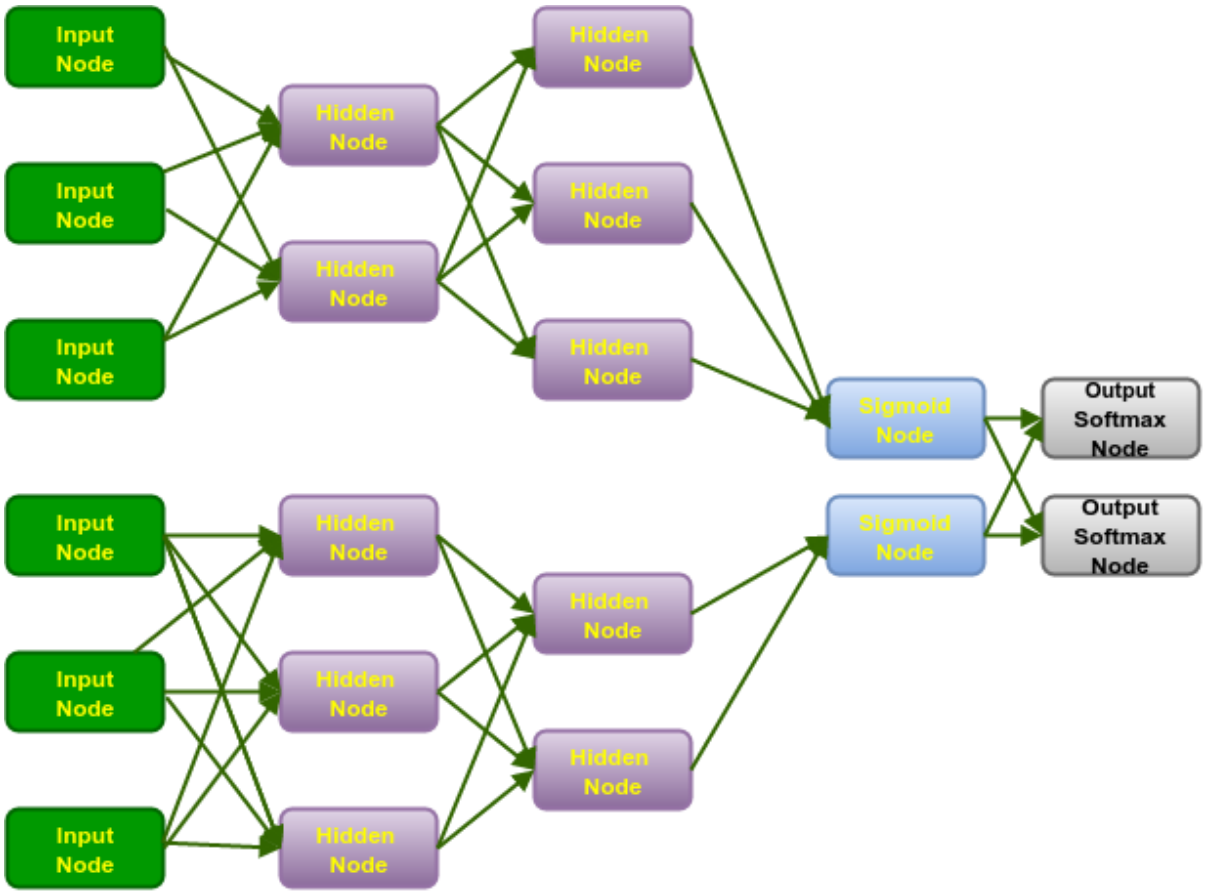

Figure 4.2: Feedforward Networks Composition with Narrow Merge

This approach allows to test the theory that hidden layers which are allowed to freely exchange distributed information from character and word level representations find phantom connections between them which allow to quickly create very specific abstractions which describe training data almost perfectly but do not generalize to unseen data very well. It is impossible to build this model using Graphlab, so the Machine Learning library was changed for this experiment. Modeling was done in *Fully Connected Edits*

Classification(Keras).ipnb IPython Notebook using Keras Deep Learning library[57] for Python. In order to minimise distortion which might have been caused by comparison of results achieved on different random slices of the same data and different Machine Learning libraries, all previously described Feedforward models were recreated in Keras and trained on the same data.

Feature Set / Model	Accuracy
Char and word level CBOW edit features / Narrow Merge	**85.4%**
Char and word level CBOW edit features / Full Merge	82.4%
Char only CBOW edit features	82.4%

Table 4.6: Feedforward Neural Network classification results for Keras testing

Table 4.6 shows results for performance evaluation of models created and tested in Keras. In order to fully explore their learning trajectory no early stop condition were determined and models were trained repeatedly until obvious overfitting. Performance was evaluated by averaging validation accuracy over 10 consecutive epochs[58] randomly chosen from the performance plateau.

This testing shows significantly better performance of the model with Narrow Merge over two other models tested on this data. This support the theory about beneficiality of prohibition of connections between levels of representation. Additional experimentation on this is needed, however, to conclude if this feature is specific to the current task or these particular data sets, or if it generalizes beyond the problem of Wikipedia edits classification.

Significant differences if performance of similarly designed Feedforward Networks between Graphlab and Keras modelling further emphasizes the need in cross-validation in order to properly estimate the best model.

4.3.5 Recurrent models

In attempt to further explore the importance of context (elements of sentence remained unchanged through editing) several different LSTM models (2.2.2.2) were constructed to process both character and word levels. As input these models take *-diff* representations of edit pairs on character and word levels respectively.

[57] https://keras.io/
[58] One epoch represents one full run over the training dataset.

To identify the role of the input word additional dimension was added to word-level and character-level embeddings at the embedding extraction step. This additional feature holds **0** if the word or the character was unchanged during editing, **1** if it was added and **-1** if it was deleted. These extended by 1 dimension features (65 for char-level and 501 for word-level) are extracted for every character and every word in the dataset. Recurrent models take these extended inputs one by one, until the end of -diff representation of a user edit. Naturally, the output of LSTM recurrent module has to have same dimensionality as its input, after recurrent processing the network finishes with standard output structure with either 2-node Softmax or 1-node Sigmoid output layer (these are interchangeable for a binary classification). Standard fully-connected hidden layers can also be added after recurrent layers. Different combinations of additional Fully-connected layers were tested, but after one layer they seem to not affect models much.

To process both Character and Word levels simultaneously structure similar to presented on the Figure 4.2 was used. In this model both LSTMs train parallely without exchanging information during recurrent training, then merged by their output layers and led to a single standard output layer. Two types of merge were tested:

1. Concatenation merge: direct outputs of LSTMs are concatenated into a single 566-dimensional vector, which is led through arbitrary number of Fully-connected (Dense) layers to the output layer.
2. Narrow Merge: similarly to the network on Figure 4.2, both recurrent outputs are shrunk into single Sigmoid-output nodes through Dense layers and these two nodes are concatenated and led directly to the output layer.

Models were trained in Keras, using the same data split as previously described Feedforward models trained in Keras. Training is done in *Edit Classification with RNNs.ipnb* IPython Notebook. Results are presented in Table 4.7. As before, models were trained without explicit stopping conditions to explore their training trajectories. Models are evaluated by averaging over 10 consecutive training epochs within the performance plateau.

Feature Set / Model	Accuracy
Full char and word level dense representations of sentences / LSTM with Concatenation Merge	84.6%
Full char and word level dense representations of sentences / LSTM with Narrow Merge	**86.1%**
Full char-only dense representation of edit sentences / LSTM	**86.1%**
Word level CBOW edit features and char-level dense representation of edit sentences / LSTM over characters Narrow merged with Feedforward over Word-level CBOW	85.8%

Table 4.7: Recurrent Neural Network classification results (Keras)

The Basic Char+Word model appears to achieve its peak of consistent accuracy after approximately 50 training epochs (final validation accuracy of 85.4% and the average of 84.6% over the 10 last epochs), and starts to succumb to overfitting after. Possibly the model can be improved with additional regularisation, but this is highly unlikely with the small number of training examples and the significant stability of this model at its peak. This recurrent model does not outperform the best Feedforward Fully-connected one.

The Char-only model with the same configuration as the char branch of the Char+Word model appears to perform better than the Char+Word model. It also overfits much slower and peaks out at 150 training epochs with a final validation accuracy of 87.1% and an average validation accuracy of 86.1% over the last 10 epochs.

The Narrow Merge Char + Word model performs at the exact same level as Char-only model. It is, however much less hectic in its training trajectory than all previous recurrent models: it maintains the performance plateau of good validation accuracy through 30 consecutive epochs. The model evaluated at **86.1%** average validation accuracy over training epochs 131-140.

Another model tested in this set was constructed from a Feedforward Network over word-level CBOW representations of user edits and LSTM network over character-level *-diff* representation of edits described here. Models were connected using Narrow Merge technique described earlier and trained simultaneously. This model was constructed in order to check the hypothesis that context inclusion benefits only character level of representation, while word-level provides more information with CBOW representation of inserted and deleted words in the edit. This assumption did not hold, as the model performed worse that the similar one with LSTMs over both character-level and word-level representations.

Overall, LSTM modeling gave promising results as an approach to incorporating context in dense sentence representations, and performed better than Feedforward Networks over the same data split. These models, however, train really slowly and strongly oscillate between epochs while trained on a dataset of this size. This makes implementation of early stopping conditions impossible in all ways but the fixed number of epochs, and as it is apparent from training trajectories of all recurrent models explored in this experiment, performance difference between consecutive epochs can reach up to 10% in validation accuracy.

Recurrent models are promising and show the best validation results from all observed models if trained for a very long time and evaluated at their best state. At the same time these models are very slow to train and tend to oscillate while trained on a small dataset, which makes them unreliable and thus they will not be considered for full 10-fold cross-validation evaluation, as they can't be used in the current state even if evaluated as the best.

4.3.6 Cross-validation and conclusions

10-fold Cross-Validation was done in order to define the best model over all explored and give the final conclusion on the usefulness of this representation of word-level data for edit classification. Cross-Validation was done in *Final Cross-Validation (Random Forests and Feedforward),ipnb* IPython Notebook. Both Keras and Graphlab were used for modelling.

To perform cross-validation, whole annotated dataset of 3637 edits was fully processed, resulting data structure was randomly reshuffled and split into 10 subsets of 363 edits. 7 edits were discarded. Each model was consecutively trained on 9 folds of data and evaluated on 1 fold, so each fold would be used for evaluation exactly once. Final result is calculated by averaging over 10 auxiliary validation results over 10 splits. Same data are used for all models in the same order.

Feature Set / Model	Accuracy
Char and word level CBOW edit features + levenshtein distance / Random Forests	**86.2%**
Char-only CBOW edit features + levenshtein distance / Random Forests	**86.2%**
Char and word level CBOW edit features / Feedforward (Dense) Neural Network	84.6%
Char-only CBOW edit features / Feedforward (Dense) Neural Network	83.1%
Char and word level CBOW edit features / Two Feedforward (Dense) Neural Networks trained with Narrow Merge	82.4%

Table 4.8: Final 10-fold Cross-Validation results

Results in Table 4.8 show that Random Forest remained the best classification algorithm despite being outperformed by Feedforward char-only network on the same validation split earlier (Table 4.5).

Results for Random forest for Char-only and Char+Word feature sets differ almost insignificantly which confirms uselessness of word-level data for Random Forest while trained on this dataset. As it was stated before in 2.2.1.4, nonparametric algorithms are completely defined by the dataset, so it is possible, that with increase in data Random Forest classifier with Char and Word-level features will start to significantly outperform Char-only one.

For this particular experiment Random Forests model trained over CBOW representations of inserted and deleted characters with addition of the levenshtein distance feature should be considered the best model, as it performs on the same level as more complex Char+Word Random Forests model and requires much less data preparation and time to train.

Cross-validation results for Feedforward Neural Networks discredited all previous assumptions about their relative performances derived from experiments on a single data split. Char+Word model performed significantly better than Char-only and the Narrow Merge model performed even worse. One possible reasoning behind that is that the cross-validation was trained with strict early stopping rules, which made more oscillating models perform worse, as more unstable models have increased chance of accidental early stopping due to series of low scores after a spice, which leads the model to be the conclusion that the performance plateau was crossed and overfitting started, and triggers the early stop.

Here it is less important, as these models still perform worse than Random Forests, and will likely not be used in the first attempt at filtering out factual edits from the corpus. Nonetheless, these results are still relevant for further exploration of methods of feature extraction and data representation for Wikipedia edit processing.

4.4 Comparison of the edits classifier performance to analogous state of the art systems

As it was mentioned in 2.3.1, the task of Wikipedia edit classification is quite specific and is not covered in published research papers that well. To my knowledge, only two papers provide direct evaluation the Edit Classification algorithms, works of Bronner and Monz [22] and Daxenberger and Gurevych [32]. Both papers explored English Wikipedia.

The best classifier by by Bronner and Monz achieved **87.14%** 10-fold cross-validation accuracy on the manually annotated dataset of comparable size. Two nonparametric classification algorithms provided the best result: Support Vector Machines and Random Forests, which is consistent with results.

Bronner and Monz used completely different approach to feature extraction, using many representational layers of linguistically-motivated features extracted by external software

stacked together. Among features used are: character-level edit distance, word-level edit distance, PoS-level edit distance, counts of PoS tags of inserted and deleted words, counts of Named Entity tags of inserted and deleted words, Acronym Recognition tags of inserted, deleted and unchanged words etc [22].

Daxenberger and Gurevych [32] constructed stylistic and factual edits classification as an auxiliary task for their multi-class Wikipedia edit classification project in order to be able to compare its performance to one reported by Bronner and Monz.

In the project they did not manually prepare the dataset and used only commented edits with revision labels as the method of class assigning. 21-label taxonomy was reduced to the binary classification problem the following way:

"Edits labeled as SPELLING/GRAMMAR, MARKUP, RELOCATION and PARAPHRASE are considered fluency edits, the remaining categories factual edits. We removed all edits labeled as OTHER, REVERT or VANDALISM from WPEC. After applying the category mapping, we deleted all edits which were labeled with both the fluency and factual category." [32, p. 585]

This method gave them a dataset of 1,262 edits, which was split into 80% of data used for training and 20% used for testing. Random Forests classifier with trees of unlimited depth was used for the experiment and performed at **90%** test accuracy. 10-fold Cross-validation was not performed. This project put a lot of focus on features external to the text and extracted from Wikipedia structure, such as metadata features and markup features.

Overall, the **86.2%** performance of the best model seems to be on par with reported performances while using features which are much easier to extract and do not require any external programs for it.

Chapter 5

Final remarks

5.1 Conclusion

This thesis has proposed an approach to automatic evaluation of the stylistic quality of natural texts through data-driven methods of Natural Language Processing. Important steps toward this goal have been implemented and tested in the presented work. Theoretical and methodological starting points for this project are data driven methods, which were applied to Wikipedia as a source for textual data mining. A program was developed for quick automatic extraction of sentences edited by users from the Wikipedia Revision History. The resulting edits have been compiled in a large-scale corpus of examples of Wikipedia editing.

Next, a need for additional refining of the resulting dataset was discussed, and number of Machine Learning classification algorithms for this task were proposed and tested. The program developed in this project was able to process approximately 10% of the whole Russian Wikipedia Revision history (200 gigabytes of textual data) in one month, resulting in the extraction of more than two millions of user edits. The best algorithm for the classification of edits into factual and stylistic ones achieved 86.2% cross-validation accuracy, which is comparable with state-of-the-art performance of similar models described in the scientific literature, using only features extracted through unsupervised learning from raw textual data. It can be concluded that the proposed method is able to produce a usable set of stylistic edits which can be further applied in future work on automatic evaluation of stylistic text quality.

5.2 Future work

The following steps towards the ultimate goal of this project could consist of using the best classifier the corpus of extracted edits; the resulting corpus should then be explored and evaluated. After that, the corpus can be used in the construction of a stylistic quality classifier for natural texts: it will take text as input and predict if it requires editing or not and return the score from the [0,1] interval on how badly it needs editing.

The classifier can be tested on Wikipedia articles from different parts of their revision trajectories for evaluation. If successful, the classifier can also be tested on the ability to generalize beyond Wikipedia articles, using Russian texts from various domains and comparing scores of the classifier to judgement scores of human evaluators.

Simultaneously, data extraction should be performed again on Wikipedia, with the goal of eventually processing the whole Russian Wikipedia Revision History. Methods of

performance improvement for Sentence Aligner suggested in 3.4.2.7 should be implemented and evaluated.

Experiments with adaptation of the processing pipeline to another language should be performed. Some experiments in that direction were already performed and indicate good results: an English Sentence Aligner was constructed using the same approach and scored 98.6% accuracy on the training set[59].

[59] Code and testing results are available in 'char2vec (En).ipynb' and 'Sentence Boundary Detection with char embeddings(english).ipynb' IPython Notebooks

Bibliography

1. Quirk, Randolph. "Towards a description of English usage." Transactions of the philological society 59.1 (1960): 40-61.
2. Wallis, Sean. "Annotation, Retrieval and Experimentation'." Annotating Variation and Change. Helsinki: Varieng,[University of Helsinki](2007).
3. Sinclair, John M. "The automatic analysis of corpora." Directions in Corpus Linguistics, Mouton de Gruyter, Berlin and New York, NY (1992): 379-397.
4. Martin, James H., and Daniel Jurafsky. "Speech and language processing."International Edition (2000).
5. Papineni, Kishore, et al. "BLEU: a method for automatic evaluation of machine translation." Proceedings of the 40th annual meeting on association for computational linguistics. Association for Computational Linguistics, 2002.
6. Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).
7. Callison-Burch, Chris, and Miles Osborne. "Re-evaluating the role of BLEU in machine translation research." In EACL. 2006.
8. Cho, Kyunghyun. "Natural Language Understanding with Distributed Representation."arXiv preprint arXiv:1511.07916 (2015).
9. Banko, Michele, and Eric Brill. "Scaling to very very large corpora for natural language disambiguation." Proceedings of the 39th annual meeting on association for computational linguistics. Association for Computational Linguistics, 2001.
10. Turney, Peter. "Mining the web for synonyms: PMI-IR versus LSA on TOEFL." (2001).
11. Meyer, Charles F., et al. "The world wide web as linguistic corpus." Language and Computers 46.1 (2003): 241-254.
12. Barbaresi, Adrien. Ad hoc and general-purpose corpus construction from web sources. Diss. ENS Lyon, 2015.
13. Bergh, Gunnar, and Eros Zanchetta. "Web linguistics." (2008): 309-327.
14. Kilgarriff, Adam. "Linguistic search engine."proceedings of Workshop on Shallow Processing of Large Corpora (SProLaC 2003). 2003.
15. Baroni, Marco, and Motoko Ueyama. "Building general-and special-purpose corpora by web crawling." Proceedings of the 13th NIJL international symposium, language corpora: Their compilation and application. 2006.
16. Spousta, Miroslav. "Web as a Corpus." WDS}'06 Proceedings of Contributed Papers. Prague. Czech Republic: Matfyzpress (2006): 179-84.
17. Mesnil, Grégoire, et al. "Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews." arXiv preprint arXiv:1412.5335 (2014).
18. Liu, Kun-Lin, Wu-Jun Li, and Minyi Guo. "Emoticon Smoothed Language Models for Twitter Sentiment Analysis." AAAI. 2012.
19. Mittal, Anshul, and Arpit Goel. "Stock prediction using twitter sentiment analysis." Stanford University, CS229 (2012).

20. Gayo-Avello, Daniel. "A meta-analysis of state-of-the-art electoral prediction from Twitter data."Social Science Computer Review (2013): 0894439313493979.
21. Schriver, Karen A. "Evaluating text quality: The continuum from text-focused to reader-focused methods." Professional Communication, IEEE Transactions on 32.4 (1989): 238-255.
22. Bronner, Amit, and Christof Monz. "User edits classification using document revision histories."Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, 2012.
23. Flesch, Rudolph. "A new readability yardstick." Journal of applied psychology 32.3 (1948): 221.
24. Graesser, Arthur C., et al. "Coh-Metrix: Analysis of text on cohesion and language." Behavior research methods, instruments, & computers 36.2 (2004): 193-202.
25. Descartes, René. Discourse on method: and other writings. Penguin Books, 1960.
26. Szegedy, Christian, et al. "Intriguing properties of neural networks." arXiv preprint arXiv:1312.6199(2013).
27. Riezler, Stefan. "On the problem of theoretical terms in empirical computational linguistics."Computational Linguistics 40.1 (2014): 235-245.
28. Sharoff, Serge. "Open-source corpora: Using the net to fish for linguistic data." International journal of corpus linguistics 11.4 (2006): 435-462.
29. Almeida, Rodrigo, Barzan Mozafari, and Junghoo Cho. "On the Evolution of Wikipedia." ICWSM. 2007.
30. Farley, B. W. A. C., and W. Clark. "Simulation of self-organizing systems by digital computer."Transactions of the IRE Professional Group on Information Theory 4.4 (1954): 76-84.
31. Max, Aurélien, and Guillaume Wisniewski. "Mining Naturally-occurring Corrections and Paraphrases from Wikipedia's Revision History." LREC. 2010.
32. Daxenberger, Johannes, and Iryna Gurevych. "Automatically Classifying Edit Categories in Wikipedia Revisions." EMNLP. 2013.
33. Bottou, Léon. "Large-scale machine learning with stochastic gradient descent." Proceedings of COMPSTAT'2010. Physica-Verlag HD, 2010. 177-186.
34. Oliver Ferschke and Torsten Zsch and Iryna Gurevych (2011). Wikipedia Revision Toolkit: Efficiently Accessing Wikipedia's Edit History. In: Proceedings of the ACL-HLT 2011 System Demonstrations.
35. Walker, Daniel J., et al. "Sentence boundary detection: A comparison of paradigms for improving MT quality." Proceedings of the MT Summit VIII. 2001.
36. Kiss, Tibor, and Jan Strunk. "Unsupervised multilingual sentence boundary detection."Computational Linguistics 32.4 (2006): 485-525.
37. Munoz, Andres. "Machine Learning and Optimization." URL: https://www. cims. nyu. edu/~ munoz/files/ml_optimization. pdf [accessed 2016-03-02][WebCite Cache ID 6fiLfZvnG] (2014).
38. Breiman, Leo. "Bagging predictors." Machine learning 24.2 (1996): 123-140.
39. Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.
40. Martin, J. Kent, and D. S. Hirschberg. "On the complexity of learning decision trees." International Symposium on Artificial Intelligence and Mathematics. 1996.

41. Goldberg, Yoav. "A primer on neural network models for natural language processing." arXiv preprint arXiv:1510.00726 (2015).
42. Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." Neural networks 2.5 (1989): 359-366.
43. Blum, Avrim L., and Ronald L. Rivest. "Training a 3-node neural network is NP-complete." Neural Networks 5.1 (1992): 117-127.
44. Raghu, Maithra, et al. "On the expressive power of deep neural networks." arXiv preprint arXiv:1606.05336 (2016).
45. Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." IEEE transactions on neural networks 5.2 (1994): 157-166.
46. Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.
47. Sparck Jones, Karen. "A statistical interpretation of term specificity and its application in retrieval." Journal of documentation 28.1 (1972): 11-21.
48. Y. Jernite, A. M. Rush, and D. Sontag. A fast variational approach for learning markov random field language models. 32nd International Conference on Machine Learning (ICML), 2015
49. Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).
50. Goldberg, Yoav, and Omer Levy. "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method." arXiv preprint arXiv:1402.3722 (2014).
51. Mikolov, T., and J. Dean. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems (2013).
52. Erhan, Dumitru, et al. "Why does unsupervised pre-training help deep learning?." Journal of Machine Learning Research 11.Feb (2010): 625-660.
53. Erhan, Dumitru, et al. "The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training." AISTATS. Vol. 5. 2009.
54. Kim, Yoon, et al. "Character-aware neural language models." arXiv preprint arXiv:1508.06615 (2015).
55. Weber, Roger, Hans-Jörg Schek, and Stephen Blott. "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces." VLDB. Vol. 98. 1998.
56. Levenshtein, Vladimir I. "Binary codes capable of correcting deletions, insertions and reversals."Soviet physics doklady. Vol. 10. 1966.
57. Segalovich, Ilya. "A Fast Morphological Algorithm with Unknown Word Guessing Induced by a Dictionary for a Web Search Engine." MLMTA. 2003.

Appendix A

Source code

Listing A.1 : WikiStreamer.java

```
import de.tudarmstadt.ukp.wikipedia.api.WikiConstants;
import de.tudarmstadt.ukp.wikipedia.parser.Paragraph;
import de.tudarmstadt.ukp.wikipedia.parser.ParsedPage;
import de.tudarmstadt.ukp.wikipedia.parser.Section;
import de.tudarmstadt.ukp.wikipedia.parser.mediawiki.FlushTemplates;
import de.tudarmstadt.ukp.wikipedia.parser.mediawiki.MediaWikiParser;
import de.tudarmstadt.ukp.wikipedia.parser.mediawiki.MediaWikiParserFactory;
import org.apache.commons.io.FileUtils;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamReader;
import java.io.*;
import java.util.*;

/**
* Hello world!
*/
public class WikiStreamer
{

  private static final String DEBUG_DIR = "/home/mithfin/Documents/wikidamps/debug/";
  private static final String OUTPUT_DIR = "/home/mithfin/anaconda2/docs/Wikiproject/Wiki/";
  private static final String DONE_IDS = "/home/mithfin/Documents/wikidamps/done.txt";
  private static final boolean PRINT_TEXTS = false;

  private static final List<Long> focus = Arrays.asList(256948L);

  public static void main(String[] args ) throws Exception
  {

    WikiStreamer wikiStreamer = new WikiStreamer();
    wikiStreamer.extractEdits(new ProgressCounter(100000, 1000));
  }

  private static final String FIRST_TEXT_ENDS = "----- first text ends -----";
  private static final String SECOND_TEXT_ENDS = "----- second text ends -----";
  private static final String COMMENT_TEXT_ENDS = "----- comment ends -----";

  private MediaWikiParser parser;
  private Process process;
```

```
  private BufferedWriter writer;

  private Set<Long> processedPages = new HashSet<>();
  private Set<Long> skipPages = new HashSet<>();

  private WikiStreamer() throws Exception {

    readDoneIds();

    MediaWikiParserFactory factory = new MediaWikiParserFactory(WikiConstants.Language.russian);
    factory.setCategoryIdentifers(Collections.singletonList("Категория"));
    factory.setDeleteTags(true);
    factory.setShowImageText(false);
    factory.setShowMathTagContent(false);
    factory.setTemplateParserClass(FlushTemplates.class);

    parser = factory.createParser();
  }

  private void readDoneIds() throws IOException {
    File doneFile = new File(DONE_IDS);
    if (doneFile.exists()) {
      for (String line : FileUtils.readLines(doneFile)) {
        skipPages.add(Long.parseLong(line));
      }
    }
//      skipPages.removeAll(focus);
  }

  private void pipePair(PageRevision oldRev, PageRevision newRev, ProgressCounter counter) throws
Exception {
    if (!Objects.equals(oldRev.id, newRev.id)) {
      return;
    }

    cleanRevisionText(oldRev);
    cleanRevisionText(newRev);

    if (null == writer) {
      startPyProcess();
    }

    writer.write(oldRev.text);
    writer.newLine();

    writer.write(FIRST_TEXT_ENDS);
    writer.newLine();

    writer.write(newRev.text);
    writer.newLine();

    writer.write(SECOND_TEXT_ENDS);
    writer.newLine();
```

```
        writer.write(newRev.comment);
        writer.newLine();

        writer.write(COMMENT_TEXT_ENDS);
        writer.newLine();

        counter.piped++;

        System.out.println("sent pair " + counter.piped);

        if (counter.batchIt()) {
            long time = System.currentTimeMillis();
            System.out.println("ending batch, processing in python starts");

            writer.close();
            process.waitFor();

            String batchResult = FileUtils.readFileToString(new File(OUTPUT_DIR + "batch.csv"));
            FileUtils.writeStringToFile(new File(OUTPUT_DIR + "Mined_edits_comments.csv"), batchResult, true);

            saveProcessed();

            System.out.println("done with batch in " + (System.currentTimeMillis() - time) / 1000 + " seconds");

            writer = null;
        }

    }

    private void cleanRevisionText(PageRevision revision) throws IOException {
        if (revision.clean) {
            return;
        }

        if (PRINT_TEXTS) {
            FileUtils.writeStringToFile(new File(DEBUG_DIR + "orig.txt"), revision.text, true);
            FileUtils.writeStringToFile(new File(DEBUG_DIR + "orig.txt"), "\n\n\n---------- " + revision.id + "  orig
page ends ---------- \n\n\n", true);
        }

        revision.text = clearSpecialSymbols(revision.text);
        revision.text = clearWikiMarkup(revision.text);
        revision.text = wikiPostProcess(revision.text);

        if (PRINT_TEXTS) {
            FileUtils.writeStringToFile(new File(DEBUG_DIR + "clean.txt"), revision.text, true);
            FileUtils.writeStringToFile(new File(DEBUG_DIR + "clean.txt"), "\n\n\n---------- " + revision.id + "
clean page ends ---------- \n\n\n", true);
        }
    }

    private void startPyProcess() throws Exception {
```

```
        ProcessBuilder processBuilder = new
    ProcessBuilder("/home/mithfin/IdeaProjects/wiki2sqlite/src/main/java/mithfin/py.sh");
        processBuilder.directory(new File("/home/mithfin/IdeaProjects/wiki2sqlite"));
        // processBuilder.redirectOutput(ProcessBuilder.Redirect.INHERIT);
        // processBuilder.redirectError(ProcessBuilder.Redirect.appendTo(new File("/dev/null")));
        processBuilder.redirectOutput(ProcessBuilder.Redirect.appendTo(new File("/dev/null")));
        processBuilder.redirectError(ProcessBuilder.Redirect.INHERIT);

        process = processBuilder.start();
        writer = new BufferedWriter(new OutputStreamWriter(process.getOutputStream()));

    }

    private static class ProgressCounter {
        int pipe;
        int piped;
        int batchSize;

        ProgressCounter(int pipe, int batchSize) {
            this.pipe = pipe;
            this.batchSize = batchSize;
        }

        boolean done() {
            return piped >= pipe;
        }

        boolean batchIt() {
            return piped > 0 && 0 == piped % batchSize;
        }
    }

    private void extractEdits(ProgressCounter counter) throws Exception {

        Long pageId = null;
        PageRevision oldRev = new PageRevision();
        PageRevision newRev = new PageRevision();

        boolean isWikiPage = false;
        boolean lookingForPageId = false;

        try (InputStream is = unzip()) {
            XMLInputFactory factory = XMLInputFactory.newInstance();

            XMLStreamReader reader = factory.createXMLStreamReader(is);

            while (reader.hasNext()) {
                int event = reader.next();

                switch (event) {
                    case XMLStreamConstants.START_ELEMENT:
                        switch (reader.getLocalName()) {
                            case "page":
                                lookingForPageId = true;
```

```
                    break;
                case "revision":
                    oldRev = newRev;
                    newRev = new PageRevision();
                    newRev.id = pageId;
                    break;
                case "text":
                    newRev.text = reader.getElementText();
                    break;
                case "id":
                    if (lookingForPageId) {
                        pageId = Long.valueOf(reader.getElementText());
                        processedPages.add(pageId);
                        lookingForPageId = false;
                    }
                    break;
                case "ns":
                    isWikiPage = "0".equals(reader.getElementText());
                    break;
                case "timestamp":
                    newRev.timestamp = reader.getElementText();
                    break;
                case "comment":
                    newRev.comment = reader.getElementText() .replace("\n", " ").replace("\r", " ");
                    break;
                case "model":
                    if (!"wikitext".equals(reader.getElementText())) {
                        newRev.wrongFormat = true;
                    }
                    break;
                case "format":
                    if (!"text/x-wiki".equals(reader.getElementText())) {
                        newRev.wrongFormat = true;
                    }
                    break;
                }
                break;

            case XMLStreamConstants.END_ELEMENT:
                String s = reader.getLocalName();
                if (s.equals("revision") && allOk(oldRev, newRev, isWikiPage)) {
                        pipePair(oldRev, newRev, counter);
                }
                break;
            }

            if (counter.done()) {
                break;
            }
        }
    }

}
```

```
  private void saveProcessed() throws IOException {
      StringBuilder procIds = new StringBuilder();
      processedPages.addAll(skipPages);
      for (Long id : processedPages) {
          procIds.append(id).append("\n");
      }
      FileUtils.writeStringToFile(new File(DONE_IDS), procIds.toString());
  }

  private boolean allOk(PageRevision oldRev, PageRevision newRev, boolean isWikiPage) {
      return isWikiPage && !skipPages.contains(oldRev.id) && !newRev.wrongFormat &&
!oldRev.wrongFormat && !isRedirect(oldRev, newRev);
  }

  private boolean isRedirect(PageRevision oldRev, PageRevision newRev) {
      String newText = newRev.text.substring(0, Math.min(200, newRev.text.length())).toLowerCase().trim();
      String oldText = oldRev.text.substring(0, Math.min(200, oldRev.text.length())).toLowerCase().trim();
      return newText.startsWith("#redirect") || oldText.startsWith("#redirect") ||
newText.startsWith("#перенаправление") || oldText.startsWith("#перенаправление");
  }

  private InputStream unzip() throws IOException {
      // install it with "sudo apt-get install p7zip-full"
      ProcessBuilder processBuilder = new ProcessBuilder("./7z.sh");
      processBuilder.directory(new File("/home/mithfin/IdeaProjects/wiki2sqlite/src/main/java/mithfin"));

      Process process = processBuilder.start();

      return process.getInputStream();
  }

  private static String clearSpecialSymbols(String tagContent) {
      return tagContent
              .replace(" ", " ")
              .replace(" ", " ")
              .replaceAll("<ref.*?</ref>\\.", ".")
              .replaceAll("<ref.*?</ref>", " ")
              .replaceAll("\\[\\[Image:.*?\\]\\]", " ")
              .replaceAll("\\[\\[Изображение:.*?\\]\\]", " ")
              .replaceAll("\\[\\[Зображення:.*?\\]\\]", " ")
              .replaceAll("\\[\\[Файл:.*?\\]\\]", " ")
              .replaceAll("\\[\\[File:.*?\\]\\]", " ")
              .replaceAll("\\[\\[bat-smg:.*?\\]\\]", " ")
              ;
  }

  private String clearWikiMarkup(String text) {

      ParsedPage page = parser.parse(text);

      if (null == page  || null == page.getSections()) return "";

      StringBuilder builder = new StringBuilder();
```

```
        for (Section section : page.getSections()) {
            for (Paragraph p : section.getParagraphs()) {
                String parText = p.getText().trim();

                if (!parText.endsWith(".") && !parText.endsWith("!") && !parText.endsWith("?")) {
                    continue;
                }

                if (parText.startsWith("== Ссылки ==")) {
                    continue;
                }

                if (parText.startsWith("==")) {
                    parText = parText.substring(2);
                    if (parText.contains("==")) {
                        parText = parText.substring(parText.indexOf("==") + 2);
                    }
                }

                builder.append(parText).append("\n");
            }
        }

        return builder.toString();
    }

    private static String wikiPostProcess(String cleanText) {
        cleanText = cleanText
                .replace("—", "-").replace("«", "\"").replaceAll("»", "\"").replace("”", "\"").replace(",,",
"\"").replace("“", "\"")
                .replaceAll("\\.\\s+\\.", ".")
                .replaceAll("\\([,-;\\s&&[^\\p{Alnum}]]*?\\)", "")
                .replaceAll("\\([,-;\\s&&[^\\p{Alnum}]]+", "(")
                .replaceAll("[,-;\\s&&[^\\p{Alnum}]]+?\\)", ")")
                .replaceAll("\\s+", " ")
        ;

        return cleanText;
    }
}
```

Listing A.2 : split_and_align.py

```
# -*- coding: utf-8 -*-
import math
import string
import pymystem3
import collections

import graphlab
import re

import sys


class Splitter:
    def __init__(self):
        self.model =
graphlab.load_model('/home/mithfin/anaconda2/docs/Wikiproject/boundary_nn_model_r7_l4
0_l10_l2')

        self.radius = len(self.model.features) / 2
        self.length = 2 * self.radius
        self.padding = []
        for i in range(self.radius):
            self.padding.append('^')

        embeddings = graphlab.SFrame.read_csv(

'/home/mithfin/anaconda2/docs/Wikiproject/char_embeddings_d150_tr1e6_w2_softmax_ada
grad_spaces.csv',
                delimiter=',', header=False, verbose=False)

        embeddings_dictionary = {}
        for i in xrange(len(embeddings)):
            vec = []
            for j in xrange(64):
                vec += [embeddings[i]['X' + str(j + 2)]]
            embeddings_dictionary[unicode(embeddings[i]['X1'], 'utf8')] = vec
        embeddings_dictionary[' '] = embeddings_dictionary['_']

        class Embeddings_Reader(dict):
            def __missing__(self, key):
                return embeddings_dictionary[u'UNK']
```

```
        self.embeddings_lookup = Embeddings_Reader(embeddings_dictionary)

    @staticmethod
    def dewikification(s):
        res = re.sub(u'[\[\{]', u'(', s)
        res = re.sub(u'[\]\}]', u')', res)
        res = re.sub(u'(\([^\n\(]*?\.)(\n{1})([^\n]*?\))', r'\1 \3', res)
        res = re.sub(u'\(править \| править вики-текст\)(\ \.)?\.?', u'.', res)
        res = re.sub(u'Информация в этом разделе устарела.\.?', u'', res)
        res = re.sub(u'Вы можете помочь проекту, обновив его и убрав после этого данный
шаблон\.?', u'', res)
        res = re.sub(u'Вы можете помочь проекту, исправив и дополнив его\.?', u'', res)
        res = re.sub(u'\r?[\s\xa0]', u' ', res)
        res = re.sub(u'[ \t]+', u' ', res)
        res = re.sub(u'[\n \r\t]*[\n\r][\n \r\t]*', u'\n', res)
        res = re.sub(u'\n+', u'\n', res)
        res = re.sub(u' +', u' ', res)
        res = re.sub(u'\(\d+\)', u'', res)
        return res

    @staticmethod
    def stop_split(text):
        dot_list = map(lambda x: x + '.', text.split('.'))
        if dot_list[-1] == '.':
            dot_list = dot_list[:-1]
        excl_list = reduce(lambda x, y: x + map(lambda z: z + '!', y.split('!')[:-1]) + [y.split('!')[-1]],
dot_list,
                           [])
        quest_list = reduce(lambda x, y: x + map(lambda z: z + '?', y.split('?')[:-1]) +
[y.split('?')[-1]], excl_list,
                            [])
        return quest_list

    @staticmethod
    def add_newline(l, stop_type):
        if stop_type == 0:
            return l + '\n'
        else:
            return l

    @staticmethod
    def remove_spaces(sent):
        sent = unicode(sent, 'utf-8')
        sent = re.sub(u'^ +', u'', sent)
        return sent.encode('utf-8')
```

```
    def split(self, input_text):

        input_text = unicode(input_text, 'utf8')

        dw_input_text = self.dewikification(input_text)
        # dw_input_text = input_text
        dw_input_list = self.stop_split(dw_input_text)
        padded_list = self.padding + dw_input_list + self.padding
        embedded_vectors = collections.deque([])

        for linenum in xrange(self.radius, len(padded_list) - self.radius):
            left = ''.join(padded_list[linenum - self.radius: linenum + 1])[:-1]
            right = ''.join(padded_list[linenum + 1: linenum + self.radius + 1])
            features = list(left)[-self.radius:] + list(right)[:self.radius]
            feature_vector = map(lambda x: self.embeddings_lookup[x], features)
            embedded_vectors.append(feature_vector)

        text_dataframe = graphlab.SFrame({'lines': dw_input_list})
        if len(text_dataframe) == 0:
            return ""

        for k in range(self.length):
            features_array = graphlab.SArray(map(lambda x: x[k], embedded_vectors))
            text_dataframe.add_column(features_array, 'feature' + str(k))

        text_dataframe['stop_type'] = self.model.classify(text_dataframe)['class']

        text_dataframe['final_lines'] = map(self.add_newline, list(text_dataframe['lines']),
                                            list(text_dataframe['stop_type']))

        final_text = reduce(lambda x, y: x + y, list(text_dataframe['final_lines']))
        final_text = map(self.remove_spaces, final_text.splitlines())

        return {"sentences": final_text}


# root_path = "./Wiki/Processed_texts/"
# root_dirs = listdir(root_path)

class Aligner:
    def __init__(self):
        self.ya_stemmer = pymystem3.mystem.Mystem()
        self.ya_stemmer.start()

    def stem(self, text):
        table = string.maketrans("", "")
```

```
        s_clean = text.translate(table, string.punctuation)
        # s_clean = re.sub(r'\r?[{0}]'.format(punctuation), '', s)
        if s_clean == '':
            return '---'
        else:
            res = ''.join(self.ya_stemmer.lemmatize(unicode(s_clean, 'utf-8')))
            ru_punctuation = u'\u2014\u2022'
            res = re.sub(u'\r?[{0}]'.format(ru_punctuation), u'', res)
            res = re.sub(u'\r? +', u' ', res)
            res = re.sub(u'\r?^ ', u'', res)
            res = re.sub(u'\r? $', u'', res)
            res = re.sub(u'\r?\n$', u'', res)
            return res

    def align(self, f1, f2):

        f1 = f1.add_row_number('number')
        f2 = f2.add_row_number('number')
        f = f1.append(f2)

        f['stemmed_sentences'] = map(lambda s: self.stem(s), list(f['sentences']))
        f['stemmed_word_count'] =
graphlab.text_analytics.count_words(f['stemmed_sentences'])
        f['stemmed_tfidf'] = graphlab.text_analytics.tf_idf(f['stemmed_word_count'])

        first_text = f[:len(f1)]
        second_text = f[len(f1):]

        tfidf_model = graphlab.nearest_neighbors.create(second_text,
features=['stemmed_tfidf'],
                                                label='number', distance='cosine')

        len1 = len(f1)
        len2 = len(f2)
        offset = abs(len2 - len1) + 10

        def round_zero(x):
            if abs(x) < 1e-6:
                return 0
            else:
                return x

        def range_decay(num1, num2, offset):
            try:
                decay = (1 + math.exp(abs(num1 - num2) - offset))
            except OverflowError:
```

```
            decay = 10e40
            pass
        return decay

    nn = tfidf_model.query(first_text, k=5, radius=0.5)

    nn['decay'] = map(lambda x, y: range_decay(x, y, offset), list(nn['query_label']),
list(nn['reference_label']))
    nn['norm_distance'] = map(lambda x, y: round_zero(x) * y, list(nn['distance']),
list(nn['decay']))

    q_l = list(nn['query_label'])
    r_l = list(nn['reference_label'])
    n_d = list(nn['norm_distance'])
    s_t = list(second_text['sentences'])

    nn_dict = {}
    st_dict = {}
    pairs = []
    try:
        aux = q_l[0]
    except IndexError:
        pass
        return graphlab.SFrame()

    for i in xrange(len(s_t)):
        st_dict[i] = s_t[i]

    for i in xrange(len(q_l)):
        if aux == q_l[i]:
            pairs.append((r_l[i], n_d[i]))
        else:
            nn_dict[aux] = dict(pairs)
            aux = q_l[i]
            pairs = [(r_l[i], n_d[i])]
        if i == len(q_l) - 1:
            nn_dict[aux] = dict(pairs)

    for i in range(len(f1)):
        try:
            nn_dict[i]
        except KeyError:
            nn_dict[i] = dict([(-1, 1)])
            pass
```

```
        alignment_list = map(lambda x: (x[0], min(x[1].items(), key=lambda k: k[1])),
nn_dict.items())

        nb_label = collections.deque()
        nb_distance = collections.deque()
        nb_text = collections.deque()
        for i in alignment_list:
            if i[1][0] == -1:
                label = -1
                distance = 1
                text = 'No match'
            else:
                label = i[1][0]
                distance = i[1][1]
                text = st_dict[label]

            nb_label.append(label)
            nb_distance.append(distance)
            nb_text.append(text)

        f_res = f1
        f_res.add_columns(graphlab.SFrame({'aligned_label': nb_label,
                                    'aligned_distance': nb_distance,
                                    'aligned_sentence': nb_text}))

        f_res = f_res[(f_res['aligned_distance'] > 0.0001) & (f_res['aligned_distance'] < 0.8)]

        return f_res

    def close(self):
        self.ya_stemmer.close()


def split_and_align(text1, text2, splitter, aligner):
    splitted1 = splitter.split(text1)
    splitted2 = splitter.split(text2)

    frame1 = graphlab.SFrame(splitted1)
    frame2 = graphlab.SFrame(splitted2)

    res = aligner.align(frame1, frame2)

    return res


# testtest = open("/home/mithfin/anaconda2/docs/Wikiproject/testtest.txt", 'r')
```

```
# input_text = testtest.read()
#
# print split(input_text)

splitter = Splitter()
aligner = Aligner()

# text1 = '#REDIRECT Балканские горы '
#
# text2 = '#перенаправление Стара-Планина '
# pairs = [(text1, text2)]


FIRST_TEXT_ENDS = "----- first text ends -----"
SECOND_TEXT_ENDS = "----- second text ends -----"
COMMENT_ENDS = "----- comment ends -----"

pairs = []
text1 = ""
text2 = ""
comment = ""
reading_second_text = False
reading_comment = False
for line in sys.stdin:
    # print line
    if FIRST_TEXT_ENDS in line:
        reading_second_text = True
        continue
    if SECOND_TEXT_ENDS in line:
        reading_second_text = False
        reading_comment = True
        continue
    if COMMENT_ENDS in line:
        reading_comment = False
        pairs.append((text1, text2, comment.strip()))
        text1 = ""
        text2 = ""
        comment = ""
        continue
    if reading_second_text:
        text2 += (" " + line)
        continue
    if reading_comment:
        comment += (" " + line)
        continue
    text1 += (" " + line)
```

```
with open('/home/mithfin/anaconda2/docs/Wikiproject/Wiki/batch.csv', 'w') as index_file:

    for pair in pairs:
        r = split_and_align(pair[0], pair[1], splitter, aligner)
        if len(r) == 0:
            continue
        sentences = list(r['sentences'])
        aligned_sentences = list(r['aligned_sentence'])
        sent_pairs = map(lambda x, y: '"' + x + '"' + ',' + '"' + y + '"' + ',' + '"' + pair[2] + '"' + '\n',
sentences, aligned_sentences)
        index_file.writelines(sent_pairs)

    index_file.close()
aligner.close()
```

Appendix B

Additional Data

Full set of extracted edits, subset of marked edits, trained character embeddings and trained sentence boundary detection models can be downloaded via following url:

https://dl.dropboxusercontent.com/u/48670620/Kotlyarov_data.7z

All IPython Notebooks, Python programs and Java programs mentioned in this paper can be downloaded here:

https://dl.dropboxusercontent.com/u/48670620/Kotlyarov_code.7z